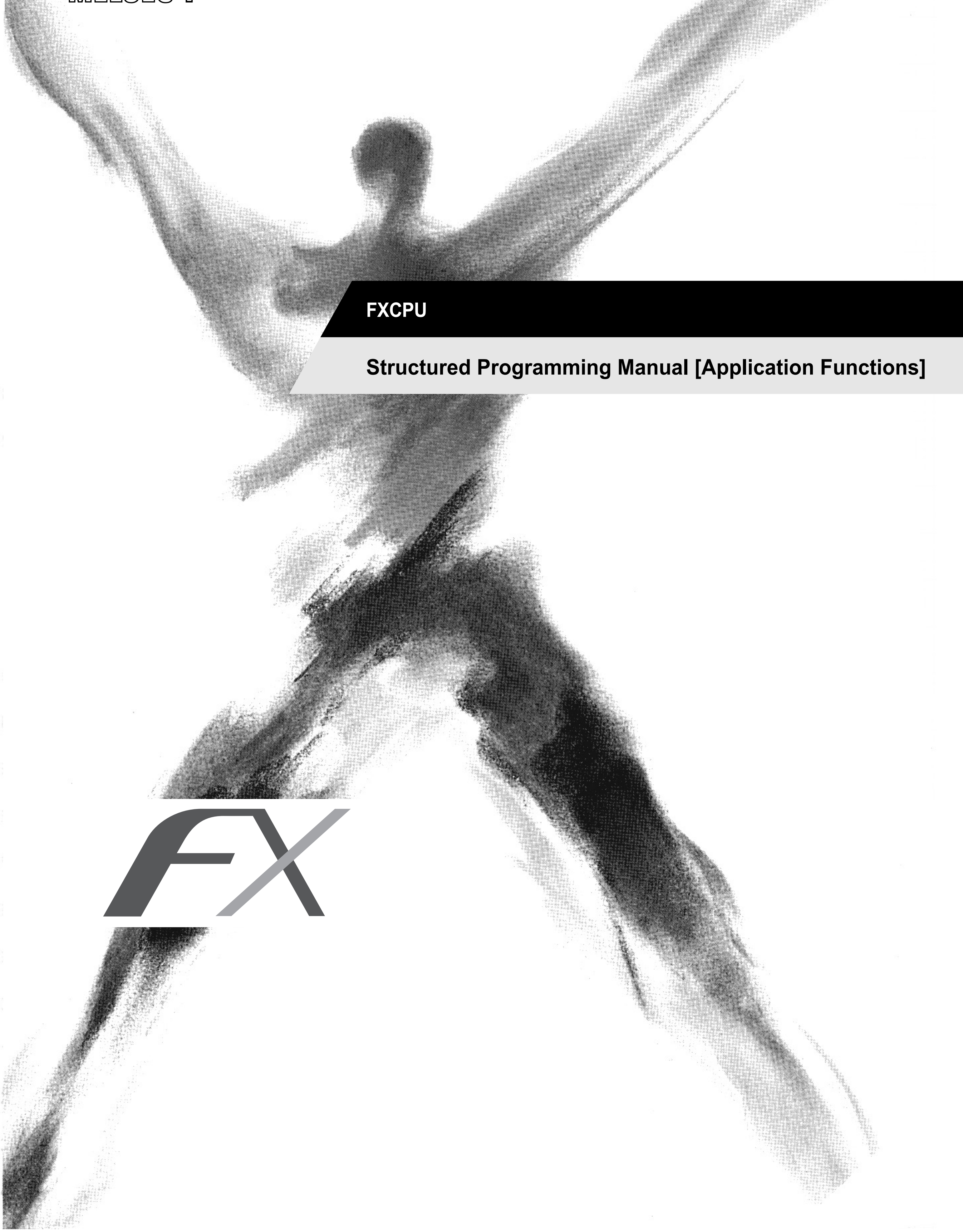
MITSUBISHI

Changes for the Better

PROGRAMMABLE CONTROLLERS

MELSEC-F



FXCPU

Structured Programming Manual [Application Functions]

FXCPU Structured Programming Manual

(Application Functions)

Manual number	JY997D34801
Manual revision	B
Date	7/2009

Foreword

This manual contains text, diagrams and explanations which will guide the reader through the safe and correct installation, use, and operation of the FX Series function for structured programs. It should be read and understood before attempting to install or use the unit.

Store this manual in a safe place so that you can take it out and read it whenever necessary. Always forward it to the end user.

This manual confers no industrial property rights or any rights of any other kind, nor does it confer any patent licenses.
Mitsubishi Electric Corporation cannot be held responsible for any problems involving industrial property rights which may occur as a result of using the contents noted in this manual.

Outline Precautions

- This manual provides information for the use of the FX Series Programmable Controllers. The manual has been written to be used by trained and competent personnel. The definition of such a person or persons is as follows;
  - a) Any engineer who is responsible for the planning, design and construction of automatic equipment using the product associated with this manual should be of a competent nature, trained and qualified to the local and national standards required to fulfill that role. These engineers should be fully aware of all aspects of safety with regards to automated equipment.
  - b) Any commissioning or service engineer must be of a competent nature, trained and qualified to the local and national standards required to fulfill that job. These engineers should also be trained in the use and maintenance of the completed product. This includes being completely familiar with all associated documentation for the said product. All maintenance should be carried out in accordance with established safety practices.
  - c) All operators of the completed equipment should be trained to use that product in a safe and coordinated manner in compliance to established safety practices. The operators should also be familiar with documentation which is connected with the actual operation of the completed equipment.

  Note: the term 'completed equipment' refers to a third party constructed device which contains or uses the product associated with this manual
- This product has been manufactured as a general-purpose part for general industries, and has not been designed or manufactured to be incorporated in a device or system used in purposes related to human life.
- Before using the product for special purposes such as nuclear power, electric power, aerospace, medicine or passenger movement vehicles, consult with Mitsubishi Electric.
- This product has been manufactured under strict quality control. However when installing the product where major accidents or losses could occur if the product fails, install appropriate backup or failsafe functions in the system.
- When combining this product with other products, please confirm the standard and the code, or regulations with which the user should follow. Moreover, please confirm the compatibility of this product to the system, machine, and apparatus with which a user is using.
- If in doubt at any stage during the installation of the product, always consult a professional electrical engineer who is qualified and trained to the local and national standards. If in doubt about the operation or use, please consult the nearest Mitsubishi Electric distributor.
- Since the examples indicated by this manual, technical bulletin, catalog, etc. are used as a reference, please use it after confirming the function and safety of the equipment and system. Mitsubishi Electric will accept no responsibility for actual use of the product based on these illustrative examples.
- This manual content, specification etc. may be changed without a notice for improvement.
- The information in this manual has been carefully checked and is believed to be accurate; however, you have noticed a doubtful point, a doubtful error, etc., please contact the nearest Mitsubishi Electric distributor.

Registration

- Microsoft® and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- CompactFlash is a trademark of SanDisk Corporation in the United States and other countries.
- The company name and the product name to be described in this manual are the registered trademarks or trademarks of each company.

Table of Contents

Positioning of This Manual ... 7
Related Manuals ... 10
Generic Names and Abbreviations Used in Manuals ... 13

1. Outline 14

1.1 Outline of Structured Programs and Programming Languages ... 14
1.1.1 Outline of structured programs ... 14
1.1.2 Programming languages ... 15
1.2 PLC Series and Programming Software Version ... 15
1.3 Cautions on Creation of Fundamental Programs ... 16
1.3.1 I/O processing and response delay ... 16
1.3.2 Double output (double coil) operation and countermeasures ... 17
1.3.3 Circuits not available in structured ladder programs and countermeasures ... 18
1.3.4 Handling of general flags ... 18
1.3.5 Handling of operation error flag ... 21

2. Function List 22

2.1 Type Conversion Functions ... 22
2.2 Standard Functions Of One Numeric Variable ... 24
2.3 Standard Arithmetic Functions ... 24
2.4 Standard Bit Shift Functions ... 25
2.5 Standard Bitwise Boolean Functions ... 25
2.6 Standard Selection Functions ... 25
2.7 Standard Comparison Functions ... 26
2.8 Standard Character String Functions ... 26
2.9 Functions Of Time Data Types ... 26
2.10 Standard Function Blocks ... 27

3. Function Construction 28

3.1 Applied Function Expression and Execution Type ... 28
3.2 Labels ... 29
3.3 Device and Address ... 32
3.4 EN and ENO ... 33

4. How to Read Explanation of Functions 34

5. Applied Functions 36

5.1 Type Conversion Functions 36
5.1.1 BOOL_TO_INT(_E) 36
5.1.2 BOOL_TO_DINT(_E) 38
5.1.3 BOOL_TO_STR(_E) 40
5.1.4 BOOL_TO_WORD(_E) 42
5.1.5 BOOL_TO_DWORD(_E) 44
5.1.6 BOOL_TO_TIME(_E) 46
5.1.7 INT_TO_DINT(_E) 48
5.1.8 DINT_TO_INT(_E) 50
5.1.9 INT_TO_BOOL(_E) 52
5.1.10 DINT_TO_BOOL(_E) 54
5.1.11 INT_TO_REAL(_E) 56
5.1.12 DINT_TO_REAL(_E) 58
5.1.13 INT_TO_STR(_E) 60
5.1.14 DINT_TO_STR(_E) 62
5.1.15 INT_TO_WORD(_E) 64
5.1.16 DINT_TO_WORD(_E) 66
5.1.17 INT_TO_DWORD(_E) 68
5.1.18 DINT_TO_DWORD(_E) 70
5.1.19 INT_TO_BCD(_E) 72
5.1.20 DINT_TO_BCD(_E) 74
5.1.21 INT_TO_TIME(_E) 76
5.1.22 DINT_TO_TIME(_E) 78
5.1.23 REAL_TO_INT(_E) 80
5.1.24 REAL_TO_DINT(_E) 82
5.1.25 REAL_TO_STR(_E) 84
5.1.26 WORD_TO_BOOL(_E) 87
5.1.27 DWORD_TO_BOOL(_E) 89
5.1.28 WORD_TO_INT(_E) 91
5.1.29 WORD_TO_DINT(_E) 93
5.1.30 DWORD_TO_INT(_E) 95
5.1.31 DWORD_TO_DINT(_E) 97
5.1.32 WORD_TO_DWORD(_E) 99
5.1.33 DWORD_TO_WORD(_E) 101
5.1.34 WORD_TO_TIME(_E) 103
5.1.35 DWORD_TO_TIME(_E) 105
5.1.36 STR_TO_BOOL(_E) 107
5.1.37 STR_TO_INT(_E) 109
5.1.38 STR_TO_DINT(_E) 111
5.1.39 STR_TO_REAL(_E) 113
5.1.40 STR_TO_TIME(_E) 116
5.1.41 BCD_TO_INT(_E) 118
5.1.42 BCD_TO_DINT(_E) 120
5.1.43 TIME_TO_BOOL(_E) 122
5.1.44 TIME_TO_INT(_E) 124
5.1.45 TIME_TO_DINT(_E) 126
5.1.46 TIME_TO_STR(_E) 128
5.1.47 TIME_TO_WORD(_E) 130
5.1.48 TIME_TO_DWORD(_E) 132

5.2 Standard Functions Of One Numeric Variable 134
5.2.1 ABS(_E) 134
5.3 Standard Arithmetic Functions 136
5.3.1 ADD_E 136
5.3.2 SUB_E 138
5.3.3 MUL_E 140
5.3.4 DIV_E 142
5.3.5 MOD(_E) 144
5.3.6 EXPT(_E) 146
5.3.7 MOVE(_E) 148
5.4 Standard Bit Shift Functions 150
5.4.1 SHL(_E) 150
5.4.2 SHR(_E) 152
5.5 Standard Bitwise Boolean Functions 154
5.5.1 AND_E 154
5.5.2 OR_E 156
5.5.3 XOR_E 158
5.5.4 NOT(_E) 160
5.6 Standard Selection Functions 162
5.6.1 SEL(_E) 162
5.6.2 MAXIMUM(_E) 164
5.6.3 MINIMUM(_E) 166
5.6.4 LIMITATION(_E) 168
5.6.5 MUX(_E) 170
5.7 Standard Comparison Functions 172
5.7.1 GT_E 172
5.7.2 GE_E 174
5.7.3 EQ_E 175
5.7.4 LE_E 176
5.7.5 LT_E 178
5.7.6 NE_E 179
5.8 Standard Character String Functions 181
5.8.1 MID(_E) 181
5.8.2 CONCAT(_E) 184
5.8.3 INSERT(_E) 187
5.8.4 DELETE(_E) 190
5.8.5 REPLACE(_E) 192
5.8.6 FIND(_E) 195
5.9 Functions Of Time Data Types 198
5.9.1 ADD_TIME(_E) 198
5.9.2 SUB_TIME(_E) 200
5.9.3 MUL_TIME(_E) 202
5.9.4 DIV_TIME(_E) 204

6. Standard Function Blocks 206

6.1 R_TRIG(_E) 206
6.2 F_TRIG(_E) 208
6.3 CTU(_E) 210
6.4 CTD(_E) 212
6.5 CTUD(_E) 214
6.6 TP(_E) 217
6.7 TON(_E) 219
6.8 TOF(_E) 221
6.9 COUNTER_FB_M 223
6.10 TIMER_10_FB_M 225
6.11 TIMER_CONT_FB_M 226
6.12 TIMER_100_FB_M 228

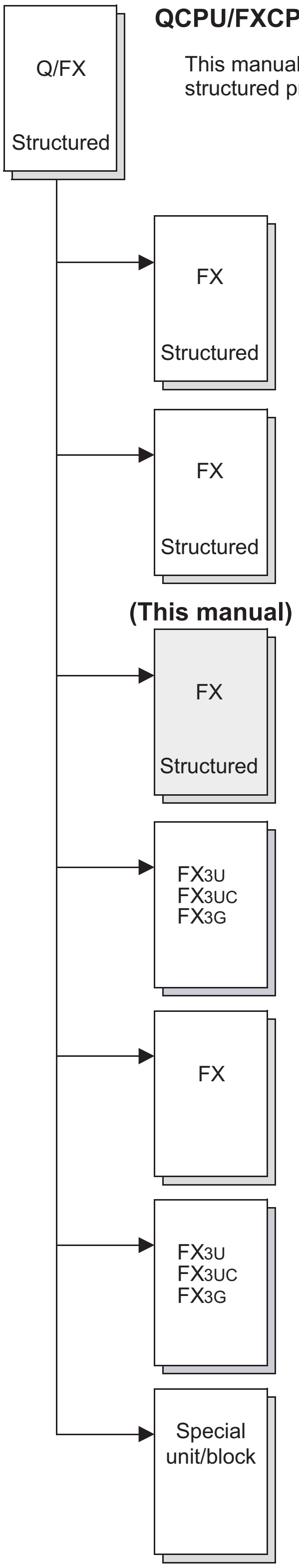
Appendix A: Correspondence between Devices and Addresses 229

Warranty 230
Revision History 231

Positioning of This Manual

This manual explains application functions for structured programs provided by GX Works2. Refer to other manuals for devices, parameters and sequence instructions.
Refer to each corresponding manual for analog, communication, positioning control and special units and blocks.

1. When using FX3U/FX3UC/FX3G PLCs



QCPU/FXCPU Structured Programming Manual (Fundamentals) (Additional Manual)

This manual explains programming methods, specifications, functions, etc. required to create structured programs.

FXCPU Structured Programming Manual (Device & Common) (Additional Manual)

This manual explains devices and parameters for structured programs provided by GX Works2.

FXCPU Structured Programming Manual (Basic & Applied Instruction)

(Additional Manual)

This manual explains sequence instructions for structured programs provided by GX Works2.

FXCPU Structured Programming Manual (Application Functions)

(Additional Manual)

This manual explains application functions for structured programs provided by GX Works2.

FX3G/FX3U/FX3UC User's Manual- Analog Control Edition (Additional Manual)

This manual explains details of analog special function blocks and analog special adapters for FX3U/FX3UC/FX3G PLCs and PID instruction.
Explanation of instructions and instructions used in program examples are expressed for GX Developer.

FX Series User's Manual -Data Communication Edition (Additional Manual)

This manual explains details of simple N:N link, parallel link, computer link, no-protocol communication (RS and RS2 instructions), programming communication and inverter communication for FX PLCs.
Explanation of instructions and instructions used in program examples are expressed for GX Developer.

FX3G/FX3U/FX3UC Series User's Manual -Positioning Edition (Additional Manual)

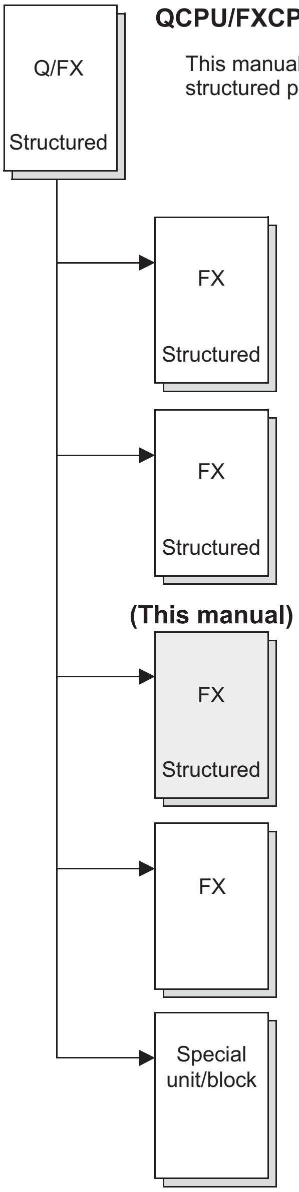
This manual explains details of wiring, instructions and operations of the positioning function built in FX3U/FX3UC/FX3G PLC main units.
Explanation of instructions and instructions used in program examples are expressed for GX Developer.

Individual manuals (Manual supplied with product or additional Manual*1)

This manual explains details of each special unit/block.
Explanation of instructions and instructions used in program examples are expressed for GX Developer.

*1. Detailed explanation may be provided by a separate manual in some products.

2. When using FX1S/FX1N/FXU/FX1NC/FX2NC PLCs



QCPU/FXCPU Structured Programming Manual (Fundamentals) (Additional Manual)

This manual explains programming methods, specifications, functions, etc. required to create structured programs.

FXCPU Structured Programming Manual (Device & Common) (Additional Manual)

This manual explains devices and parameters for structured programs provided by GX Works2.

FXCPU Structured Programming Manual (Basic & Applied Instruction) (Additional Manual)

This manual explains sequence instructions for structured programs provided by GX Works2.

FXCPU Structured Programming Manual (Application Functions) (Additional Manual)

This manual explains application functions for structured programs provided by GX Works2.

FX Series User's Manual -Data Communication Edition (Additional Manual)

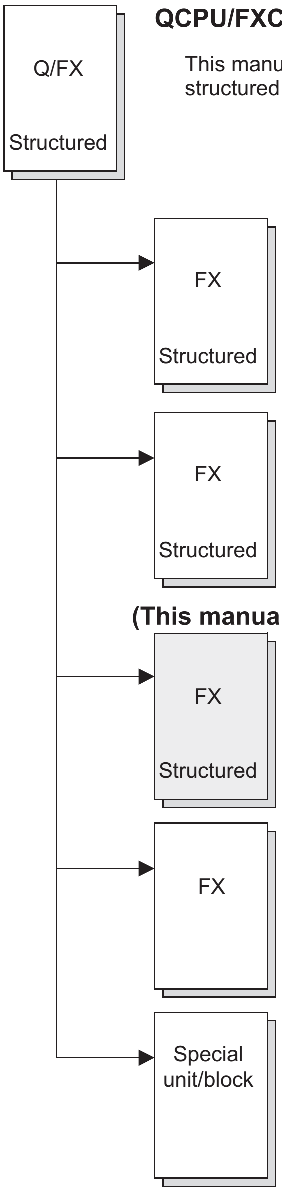
This manual explains details of simple N:N link, parallel link, computer link, no-protocol communication (RS instruction), programming communication and inverter communication for FX PLCs.
Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

Individual manuals (Manual supplied with product or additional Manual*1)

This manual explains details of each special unit/block.
Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

*1. Detailed explanation may be provided by a separate manual in some products.

3. When using FX0/FX0S/FX0N/FXU/FX2C PLCs



QCPU/FXCPU Structured Programming Manual (Fundamentals) (Additional Manual)

This manual explains programming methods, specifications, functions, etc. required to create structured programs.

FXCPU Structured Programming Manual (Device & Common) (Additional Manual)

This manual explains devices and parameters for structured programs provided by GX Works2.

FXCPU Structured Programming Manual (Basic & Applied Instruction) (Additional Manual)

This manual explains sequence instructions for structured programs provided by GX Works2.

(This manual)

FXCPU Structured Programming Manual (Application Functions) (Additional Manual)

This manual explains application functions for structured programs provided by GX Works2.

FX Series User's Manual -Data Communication Edition (Additional Manual)

This manual explains details of parallel link, computer link, no-protocol communication (RS instruction) and programming communication for FX PLCs.
Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

Individual manuals (Manual supplied with product or additional Manual [*1])

This manual explains details of each special unit/block.
Explanation of instructions and instructions used in program examples are expressed for GX Developer and FX-PCS/WIN.

*1. Detailed explanation may be provided by a separate manual in some products.

Related Manuals

This manual explains devices and parameters for structured programs provided by GX Works2.
Refer to other manuals for sequence instructions and applied functions.
This chapter introduces only reference manuals for this manual and manuals which describe the hardware information of PLC main units.
Manuals not introduced here may be required in some applications.
Refer to the manual of the used PLC main unit and manuals supplied together with used products.
Contact the distributor for acquiring required manuals.

Common among FX PLCs [structured]

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
QCPU/FXCPU Structured Programming Manual (Fundamentals)	SH-080782	Additional Manual	Programming methods, specifications, functions, etc. required to create structured programs	13JW06
FXCPU Structured Programming Manual (Device & Common)	JY997D26001	Additional Manual	Devices, parameters, etc. provided in structured projects of GX Works2	09R920
FXCPU Structured Programming Manual (Basic & Applied Instruction)	JY997D34701	Additional Manual	Sequence instructions provided in structured projects of GX Works2	09R921
FXCPU Structured Programming Manual (Application Functions)	JY997D34801	Additional Manual	Application functions provided in structured projects of GX Works2	09R922

FX3U/FX3UC/FX3G PLCs

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX3U Series Hardware Manual	JY997D18801	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3U extracted from the FX3U Series User's Manual - Hardware Edition. For detailed explanation, refer to the FX3U Series User's Manual - Hardware Edition.	-
FX3U Series User's Manual- Hardware Edition	JY997D16501	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3U PLC main unit.	09R516
FX3UC (D, DSS) Series Hardware Manual	JY997D28601	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3UC (D, DSS) extracted from the FX3UC Series User's Manual - Hardware Edition. For detailed explanation, refer to the FX3UC Series User's Manual - Hardware Edition.	-
FX3UC-32MT-LT-2 Hardware Manual	JY997D31601	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3UC-32MT-LT-2 extracted from the FX3UC Series User's Manual - Hardware Edition. For detailed explanation, refer to the FX3UC Series User's Manual - Hardware Edition.	-
FX3UC Series User's Manual - Hardware Edition	JY997D28701	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3UC PLC main unit.	09R519
FX3G Series Hardware Manual	JY997D33401	Supplied with product	I/O specifications, wiring and installation of the PLC main unit FX3G extracted from the FX3G Series User's Manual - Hardware Edition. For detailed explanation, refer to the FX3G Series User's Manual - Hardware Edition.	-
FX3G Series User's Manual- Hardware Edition	JY997D31301	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX3G PLC main unit.	09R521

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
Programming				
FX3G/FX3U/FX3UC User's Manual-Analog Control Edition	JY997D16701	Additional Manual	Detaileds about the analog special function block (FX3U-4AD, FX3U-4DA, FX3UC-4AD) and analog special adapter (FX3U-****-ADP).	09R619
FX Series User's Manual -Data Communication Edition	JY997D16901	Additional Manual	Details about simple N : N link, parallel link, computer link and no-protocol communication (RS instruction and FX2N-232IF).	09R715
FX3G/FX3U/FX3UC Series User's Manual -Positioning Edition	JY997D16801	Additional Manual	Details about the positioning function built in the FX3G/FX3U/FX3UC Series.	09R620
FX3U-CF-ADP User's Manual	JY997D35401	Additional Manual	Describes details of the FX3U-CF-ADP CF card special adapter.	09R720

FX1S/FX1N/FX2N/FX1NC/FX2NC PLCs

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX1S HARDWARE MANUAL	JY992D83901	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1S PLC main unit.	-
FX1N HARDWARE MANUAL	JY992D89301	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1N PLC main unit.	-
FX2N HARDWARE MANUAL	JY992D66301	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX2N PLC main unit.	09R508
FX1NC HARDWARE MANUAL	JY992D92101	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX1NC PLC main unit. (Japanese only)	09R505
FX2NC HARDWARE MANUAL	JY992D76401	Additional Manual	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX2NC PLC main unit.	09R509
Programming				
FX Series User's Manual -Data Communication Edition	JY997D16901	Additional Manual	Details about simple N : N link, parallel link, computer link and no-protocol communication (RS instruction and FX2N-232IF).	09R715

FX0/FX0S/FX0N/FXU/FX2C PLCs [whose production is finished]

Manual name	Manual number	Supplied with product or Additional Manual	Contents	Model name code
PLC main unit				
FX0/FX0N HARDWARE MANUAL	JY992D47501	Supplied with product	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX0/FX0N PLC main unit.	-
FX0S HARDWARE MANUAL	JY992D55301	Supplied with product	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FX0S PLC main unit.	-
FX/FX2C HARDWARE MANUAL	JY992D47401	Supplied with product	Details about the hardware including I/O specifications, wiring, installation and maintenance of the FXU/FX2C PLC main unit.	-
Programming				
FX Series User's Manual -Data Communication Edition	JY997D16901	Additional Manual	Details about simple N : N link, parallel link, computer link and no-protocol communication (RS instruction and FX2N-232IF).	09R715

Manuals of models whose production is finished

Production is finished for FX0/FX0S/FX0N/FXU/FX2C PLCs.

Generic Names and Abbreviations Used in Manuals

Abbreviation/generic name	Name
PLCs	
FX3U Series or FX3U PLC	Generic name of FX3U Series PLCs
FX3UC Series or FX3UC PLC	Generic name of FX3UC Series PLCs
FX3G Series or FX3G PLC	Generic name of FX3G Series PLCs
FX2N Series or FX2N PLC	Generic name of FX2N Series PLCs
FX2NC Series or FX2NC PLC	Generic name of FX2NC Series PLCs
FX1N Series or FX1N PLC	Generic name of FX1N Series PLCs
FX1NC Series or FX1NC PLC	Generic name of FX1NC Series PLCs These products can only used in Japan.
FX1S Series or FX1S PLC	Generic name of FX1S Series PLCs
FXU Series or FXU PLC	Generic name of FXU(FX,FX2) Series PLCs
FX2C Series or FX2C PLC	Generic name of FX2C Series PLCs
FX0N Series or FX0N PLC	Generic name of FX0N Series PLCs
FX0S Series or FX0S PLC	Generic name of FX0S Series PLCs
FX0 Series or FX0 PLC	Generic name of FX0 Series PLCs
Special adapters	
CF card special adapter	Generic name of CF card special adapters
CF-ADP	FX3U-CF-ADP
Programming language	
ST	Abbreviation of structured text language
Structured ladder	Abbreviation of ladder diagram language
Manuals	
Q/FX Structured Programming Manual (Fundamentals)	Abbreviation of QCPU/FXCPU Structured Programming Manual (Fundamentals)
FX Structured Programming Manual (Device & Common)	Abbreviation of FXCPU Structured Programming Manual (Device & Common)
FX Structured Programming Manual (Basic & Applied Instruction)	Abbreviation of FXCPU Structured Programming Manual (Basic & Applied Instruction)
FX Structured Programming Manual (Application Functions)	Abbreviation of FXCPU Structured Programming Manual (Application Functions)
COMMUNICATION CONTROL EDITION	Abbreviation of FX Series User's Manual-DATA COMMUNICATION CONTROL EDITION
ANALOG CONTROL EDITION	Abbreviation of FX3G/FX3U/FX3UC Series User's Manual-ANALOG CONTROL EDITION
POSITIONING CONTROL EDITION	Abbreviation of FX3G/FX3U/FX3UC Series User's Manual-POSITIONING CONTROL EDITION

1. Outline

This manual explains applied functions for structured programs provided by GX Works2.
Refer to a different manual for devices, parameters and sequence instructions.
Refer to the following manual for labels, data types and programming languages for structured programs:
→ **Q/FX Structured Programming Manual (Fundamentals)**

1.1 Outline of Structured Programs and Programming Languages

1.1.1 Outline of structured programs

You can construct two or more programs (program blocks) into one program.
Because you can divide the entire machine processing into small sub processes and create a program for each sub process, you can efficiently create a program for a large system.

1. **Structured program**
   Program structuring is a technique to divide the contents of control executed by the PLC CPU into hierarchical small units (blocks) of processing, and then construct a program. By using this technique, you can design a program while recognizing structuring of a sequence program.

   Advantages of hierarchical program

   - You can examine the outline of a program at first, and then design its details gradually.
   - Program blocks located at the lowest level in the hierarchy are extremely simple and highly independent.

   Advantages of program consisting of program blocks

   - Because the processing of each block is clear, the entire program is easy to understand.
   - The entire program can be divided into several blocks that are created by several people.
   - The program reusability is improved, and the development efficiency is improved accordingly.

2. **Improved reusability of programs**
   You can save program blocks in a library. Program resources in the library can be shared, and often used again.

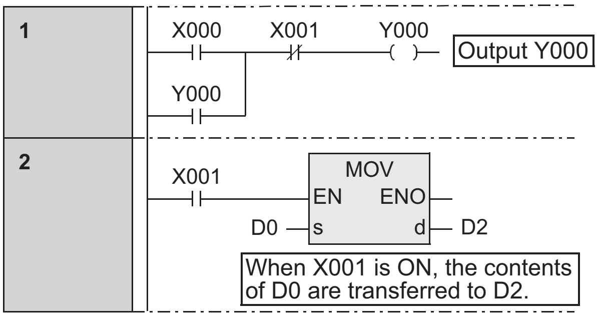
1.1.2 Programming languages

The following programming languages can be used in each program block.

Graphic languages

1. Structured ladder language

This graphic language is created based on the relay circuit design technology.
Any circuit always starts from the bus line located on the leftmost.
The structured ladder language consists of contacts, coils, functions and function blocks. These components are connected with vertical lines and horizontal lines.



Text language

1. ST (Structured text) language

The ST language can describe control achieved by syntax using selective branches with conditional statements and repetition by repetitive statements in the same way as high-level languages such as C language.
By using the ST language, you can create simple programs easy to understand.

```
Y000:=(X000 OR Y000) AND NOT X001;
IF X001 THEN
    D2:=D0; (When X001 is ON, the contents of D0 are transferred to D2.)
END_IF;
IF X002 THEN
    D4:=D4+1; (When X002 is ON, the contents of D4 are added by "1".)
ELSE
    D6:=D6+1; (When X002 is OFF, the contents of D6 are added by "1".)
END_IF;
```

1.2 PLC Series and Programming Software Version

PLC Series	Software package name (model name)	GX Works2 version
FX3U•FX3UC	GX Works2 (SW1DNC-GXW2-E)	Ver. 1.08J or later
FX3G		
FX2N•FX2NC		
FX1N•FX1NC		
FX1S		
FXU/FX2C		
FX0N		
FX0•FX0S		

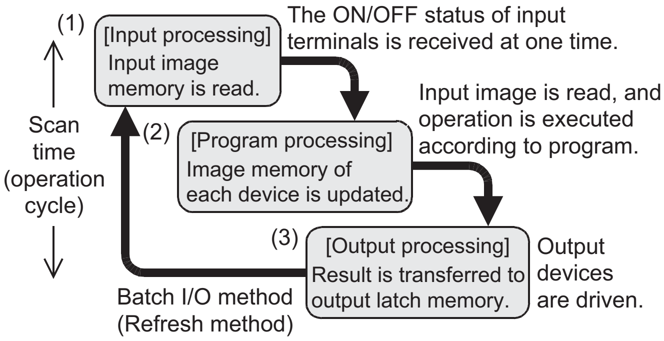
1.3 Cautions on Creation of Fundamental Programs

This section explains cautions on programming.
Refer to the following manual for cautions on structured programs and programming languages:
→ **Q/FX Structured Programming Manual (Fundamentals)**
Refer to the following programming manual for detailed operations of and cautions on devices and parameters:
→ **FX Structured Programming Manual (Device & Common)**

1.3.1 I/O processing and response delay

1. Operation timing of I/O relays and response delay

FX PLCs execute the I/O processing by repeating the processing (1) to processing (3). Accordingly, the control executed by PLCs contains not only the drive time of input filters and output devices but also the response delay caused by the operation cycle.


(1)
[Input processing]
Input image memory is read.
The ON/OFF status of input terminals is received at one time.
Scan time (operation cycle)
(2)
[Program processing]
Image memory of each device is updated.
Input image is read, and operation is executed according to program.
(3)
[Output processing]
Result is transferred to output latch memory.
Output devices are driven.
Batch I/O method (Refresh method)

Acquiring the latest I/O information

For acquiring the latest input information or immediately outputting the operation result in the middle of the operation cycle shown above, the I/O refresh instruction (REF) is available.

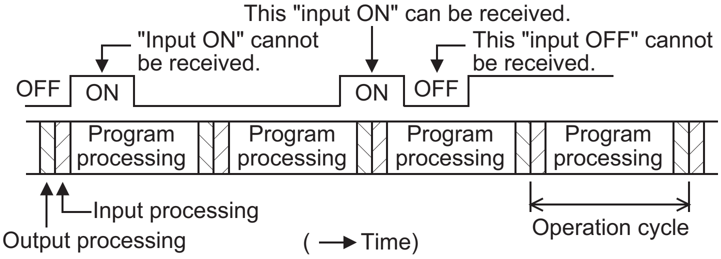
2. Short pulses cannot be received.

The ON duration and OFF duration of inputs in PLCs require longer time than "PLC cycle time + Input filter response delay".
When the response delay "10 ms" of the input filter is considered and the cycle time is supposed as "10 ms", the ON duration and OFF duration should be at least 20 ms respectively.
Accordingly, PLCs cannot handle input pulses at 25 Hz (= 1000 /(20 + 20)) or more. However, the situation can be improved by PLC special functions and applied instructions.

Convenient functions for improvement

By using the following functions, PLCs can receive pulses shorter than the operation cycle:

- High speed counter function
- Input interrupt function
- Pulse catch function
- Input filter value adjustment function



1.3.2 Double output (double coil) operation and countermeasures

This subsection explains the double output (double coil) operation and countermeasures.

1. Operation of double outputs

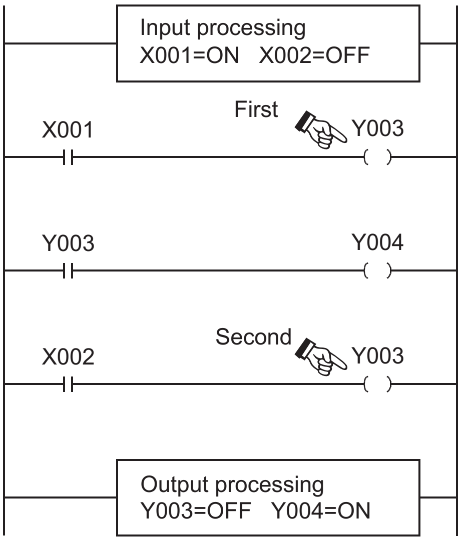
When a coil (output variable) is used twice (double coils) in another program block to be executed or in the same program block, the PLC gives priority to the latter coil.

Suppose that the same coil Y003 is used in two positions as shown in the right figure.
For example, suppose that X001 is ON and X002 is OFF.

In the first coil Y003, the image memory is set to ON and the output Y004 is also set to ON because the input X001 is ON.

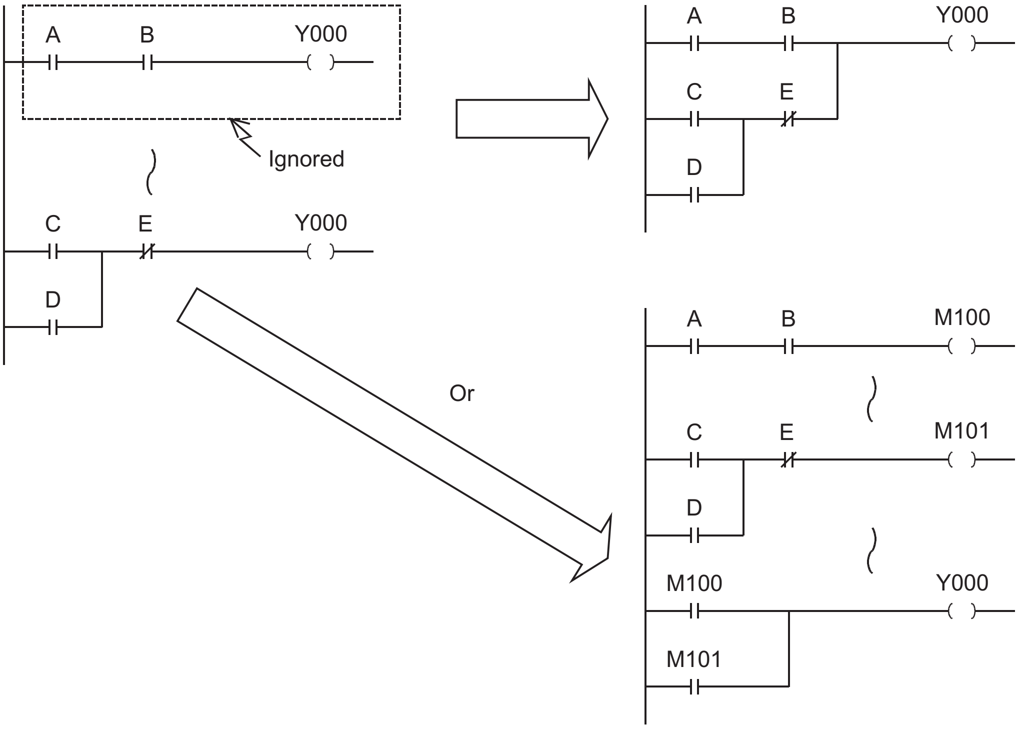
In the second coil Y003, however, the image memory is set to OFF because the input X002 is OFF.

As a result, the actual output to the outside is "Y003: OFF, Y004: ON".



2. Countermeasures against double outputs

Double outputs (double coils) do not cause an illegal input error (program error), but the operation is complicated as described above.
Change the program as shown in the example below.

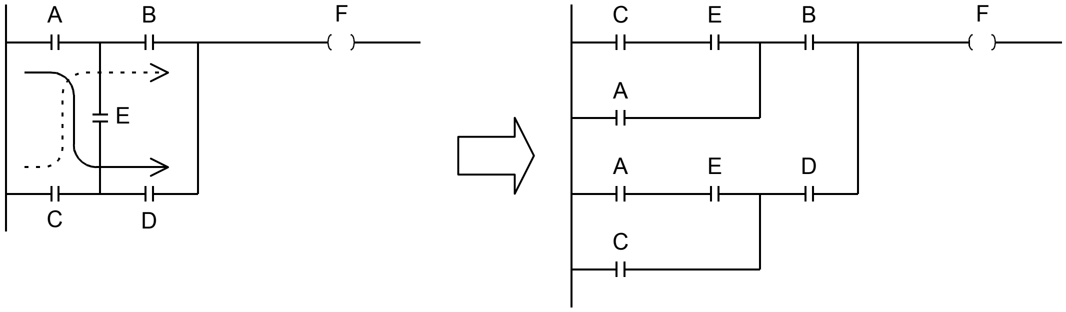


The SET and RST instructions or jump instruction can be used instead, or a same output coil can be programmed at each state using step ladder instructions STL and RET.
When you use the step ladder instructions STL and RET, note that the PLC regards it as double coils if you program, inside the state, an output coil located outside the RET instruction from another program block or the STL instruction.

1.3.3 Circuits not available in structured ladder programs and countermeasures

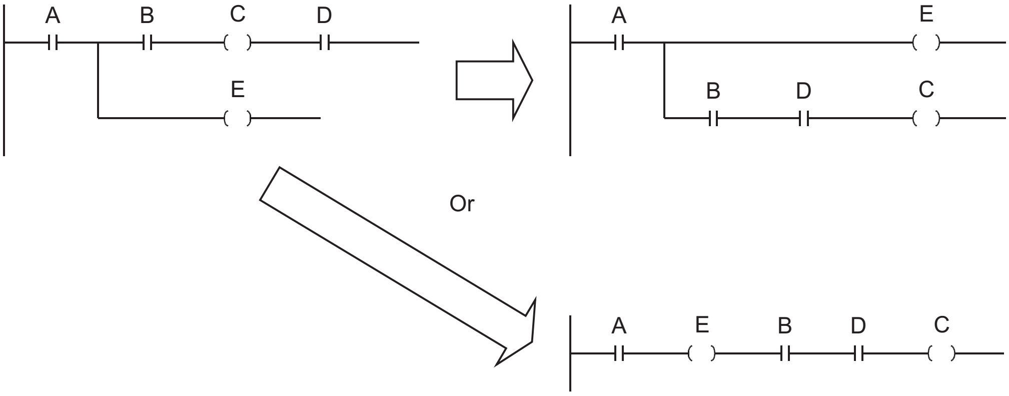
1. Bridge circuit

A circuit in which the current flows in both directions should be changed as shown in the right figure (so that a circuit without D and a circuit without B are connected in parallel).



2. Coil connection position

- You can program a contact on the right side of a coil. In this case, make sure to program a coil (including a function or function block) at the end of the circuit.



1.3.4 Handling of general flags

The following flags are valid in general sequence instructions:
(Examples)

M8020:Zero flag　M8021:Borrow flag　M8022:Carry flag
M8029:Instruction execution complete flag　M8090:Block comparison signal*1
M8328:Instruction non-execution flag*1　M8329:Instruction execution abnormal complete flag*2
M8304:Zero flag*1　M8306:Carry flag*1

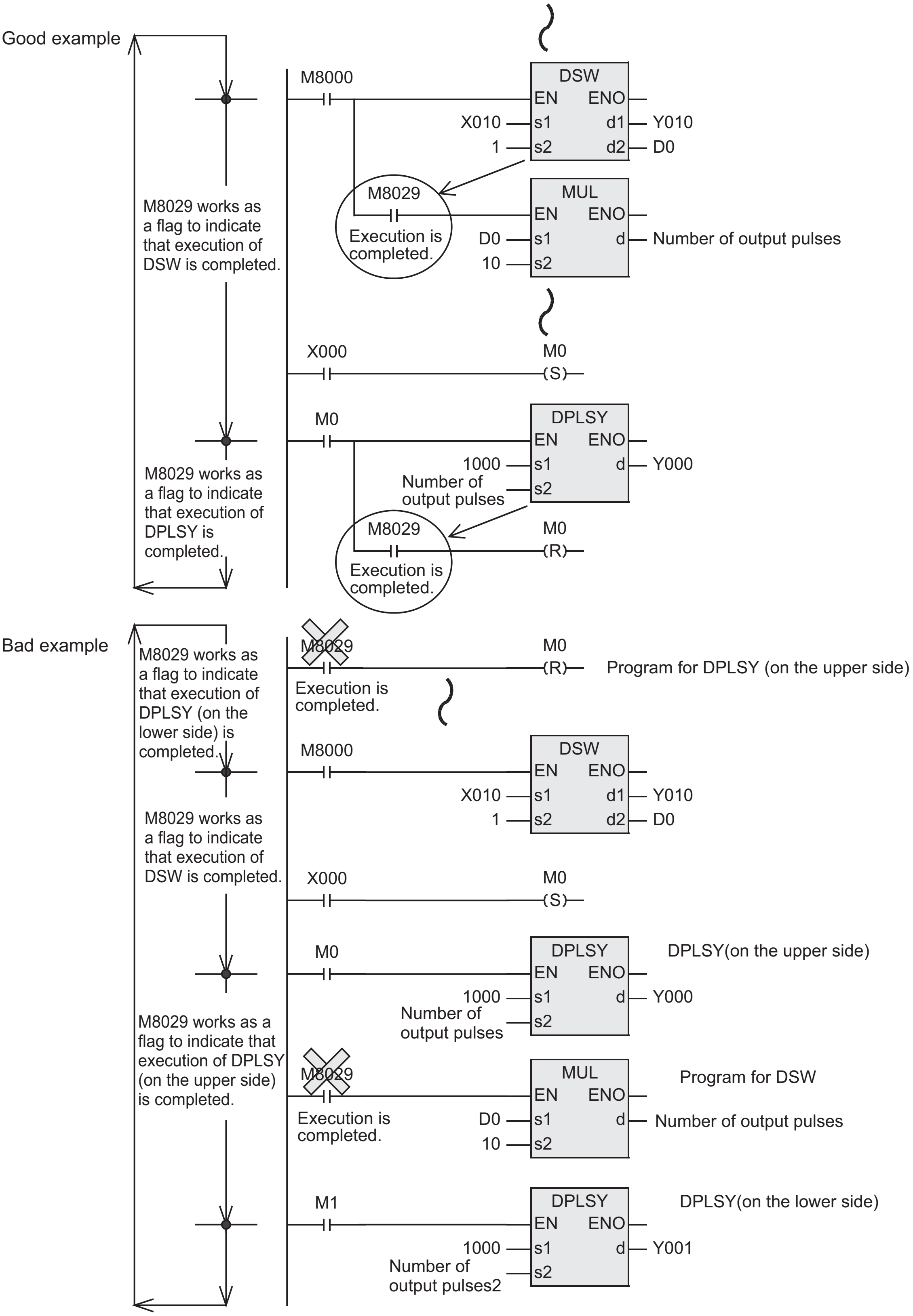
*1. Supported only in FX3U/FX3UC PLCs.

*2. Supported only in FX3U/FX3UC/FX3G PLCs.

Each of these flags turns ON or OFF every time the PLC executes a corresponding instruction. These flags do not turn ON or OFF when the PLC does not execute a corresponding instruction or when an error occurs. Because these flags are related to many sequence instructions, their ON/OFF status changes every time the PLC executes each corresponding instruction.
Refer to examples in the next page, and program a flag contact just under the target sequence instruction.

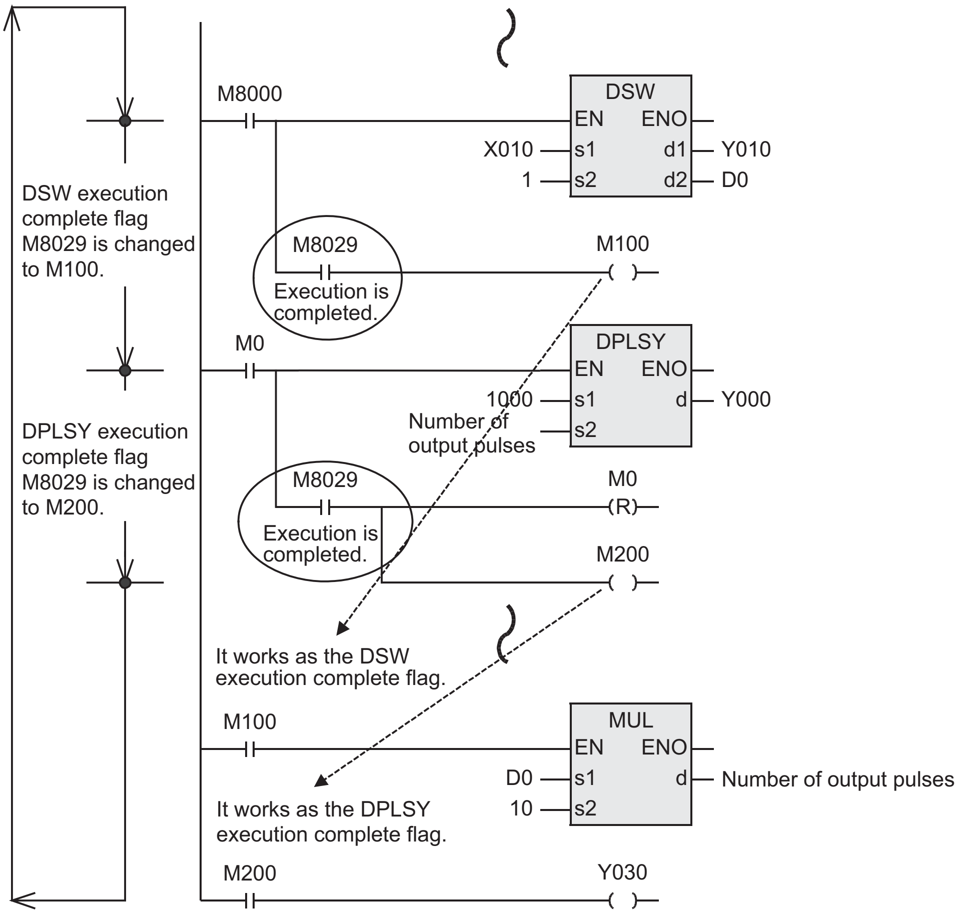
1. Program containing many flags (Example of instruction execution complete flag M8029)

If you program the instruction execution complete flag M8029 twice or more together for two or more sequence instructions which actuate the flag M8029, you cannot judge easily by which sequence instruction the flag M8029 is controlled. In addition, the flag M8029 does not turn ON or OFF correctly for each corresponding sequence instruction.
Refer to the next page when you would like to use the flag M8029 in any position other than the position just under the corresponding sequence instruction.


Good example
M8000
DSW
EN ENO
X010 s1 d1 Y010
1 s2 d2 D0
M8029
Execution is completed.
MUL
EN ENO
D0 s1 d Number of output pulses
10 s2
M8029 works as a flag to indicate that execution of DSW is completed.
X000
M0
(S)
M0
DPLSY
EN ENO
1000 s1 d Y000
Number of output pulses s2
M8029 works as a flag to indicate that execution of DPLSY is completed.
M8029
Execution is completed.
M0
(R)
Bad example
M8029 works as a flag to indicate that execution of DPLSY (on the lower side) is completed.
M8029
Execution is completed.
M0
(R) Program for DPLSY (on the upper side)
M8000
DSW
EN ENO
X010 s1 d1 Y010
1 s2 d2 D0
M8029 works as a flag to indicate that execution of DSW is completed.
X000
M0
(S)
M0
DPLSY
EN ENO
DPLSY(on the upper side)
1000 s1 d Y000
Number of output pulses s2
M8029 works as a flag to indicate that execution of DPLSY (on the upper side) is completed.
M8029
MUL
EN ENO
Program for DSW
Execution is completed.
D0 s1 d Number of output pulses
10 s2
M1
DPLSY
EN ENO
DPLSY(on the lower side)
1000 s1 d Y001
Number of output pulses2 s2

2. Introduction of a method to use flags in any positions other than positions just under sequence instructions

If two or more sequence instructions are programmed, general flags turn ON or OFF when each corresponding instruction is executed.
Accordingly, when using a general flag in any position other than a position just under a sequence instruction, set to ON or OFF another device (variable) just under the sequence instruction, and then use the contact of such device (variable) as the command contact.



1.3.5 Handling of operation error flag

When there is an error in the instruction construction, target device or target device number range and an error occurs while operation is executed, the following flag turns ON and the error information is stored.

1. Operation error

Error flag	Device which stores error code	Device which stores error occurrence step	
		FX0/FX0S/FX0N/FXU/FX2C/FX1S /FX1N/FX2N/FX1NC/FX2NC/FX3G	FX3U/FX3UC
M8067	D8067	D8069*1	D8315, D8314

*1. When an error occurs in a step up to the step No. 32767 in the FX3U/FX3UC PLC, you can check the error occurrence step also in D8069 (16 bits).

- When an operation error has occurred, M8067 turns ON, D8067 stores the operation error code, and the specified device (shown in the table above) stores the error occurrence step.
- When another error occurs in another step, the stored data is updated in turn to the error code and step number of the new error. (These devices are set to OFF when errors are cleared.)
- When the PLC mode changes from STOP to RUN, these devices are cleared instantaneously, and then turn ON again if errors have not been cleared.

2. Operation error latch

Error flag	Device which stores error code	Device which stores error occurrence step	
		FX0/FX0S/FX0N/FXU/FX2C/FX1S /FX1N/FX2N/FX1NC/FX2NC/FX3G	FX3U/FX3UC
M8068	-	D8068*2	D8313, D8312

*2. When an error occurs in a step up to the step No. 32767 in the FX3U/FX3UC PLC, you can check the error occurrence step also in D8068 (16 bits).

- When an operation error has occurred, M8068 turns ON, and the device shown in the table above stores the error occurrence step.
- Even if another error occurs in another step, the stored data is not updated and remains held until these devices are forcibly set to OFF or until the power is turned OFF.

2. Function List

This chapter introduces a list of functions available in programming.

2.1 Type Conversion Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
BOOL_TO_INT(_E)	Converts bit data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.1
BOOL_TO_DINT(_E)	Converts bit data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.2
BOOL_TO_STR(_E)	Converts bit data into string data.	✓								Subsection 5.1.3
BOOL_TO_WORD(_E)	Converts bit data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.4
BOOL_TO_DWORD (_E)	Converts bit data into double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.5
BOOL_TO_TIME(_E)	Converts bit data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.6
INT_TO_DINT(_E)	Converts word [signed] data into double word [signed] data	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.7
DINT_TO_INT(_E)	Converts double word [signed] data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.8
INT_TO_BOOL(_E)	Converts word [signed] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.9
DINT_TO_BOOL(_E)	Converts double word [signed] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.10
INT_TO_REAL(_E)	Converts word [signed] data into float (single precision) data.	✓	*1	✓						Subsection 5.1.11
DINT_TO_REAL(_E)	Converts double word [signed] data into float (single precision) data.	✓	*1	✓						Subsection 5.1.12
INT_TO_STR(_E)	Converts word [signed] data into string data.	✓								Subsection 5.1.13
DINT_TO_STR(_E)	Converts double word [signed] data into string data.	✓								Subsection 5.1.14
INT_TO_WORD(_E)	Converts word [signed] data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.15
DINT_TO_WORD(_E)	Converts double word [signed] data into word [unsigned]/bit string [16-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.16
INT_TO_DWORD(_E)	Converts word [signed] data into double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.17
DINT_TO_DWORD (_E)	Converts double word [signed] data into double word [unsigned]/bit string[32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.18
INT_TO_BCD(_E)	Converts word [signed] data into BCD data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.19
DINT_TO_BCD(_E)	Converts double word [signed] data into BCD data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.20
INT_TO_TIME(_E)	Converts word [signed] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.21
DINT_TO_TIME(_E)	Converts double word [signed] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.22

*1. The function is provided in the FX3G Series Ver.1.10 or later.

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
REAL_TO_INT(_E)	Converts float (single precision) data into word [signed] data.	✓	*1	✓						Subsection 5.1.23
REAL_TO_DINT(_E)	Converts float (single precision) data into double word [signed] data.	✓	*1	✓						Subsection 5.1.24
REAL_TO_STR(_E)	Converts float (single precision) data into string data.	✓								Subsection 5.1.25
WORD_TO_BOOL(_E)	Converts word [unsigned]/bit string [16-bit] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.26
DWORD_TO_BOOL (_E)	Converts double word [unsigned]/bit string [32-bit] data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.27
WORD_TO_INT(_E)	Converts word [unsigned]/bit string [16-bit] data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.28
WORD_TO_DINT(_E)	Converts word [unsigned]/bit string [16-bit] data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.29
DWORD_TO_INT(_E)	Converts double word [unsigned]/bit string [32-bit] data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.30
DWORD_TO_DINT (_E)	Converts double word [unsigned]/bit string [32-bit] data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.31
WORD_TO_DWORD (_E)	Converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit].	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.32
DWORD_TO_WORD (_E)	Converts double word [unsigned]/bit string [32-bit] data into word [unsigned]/bit string [16-bit]data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.33
WORD_TO_TIME(_E)	Converts word [unsigned]/bit string [16-bit] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.34
DWORD_TO_TIME (_E)	Converts double word [unsigned]/bit string [32-bit] data into time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.35
STR_TO_BOOL(_E)	Converts string data into bit data.	✓								Subsection 5.1.36
STR_TO_INT(_E)	Converts string data into word [signed] data.	✓								Subsection 5.1.37
STR_TO_DINT(_E)	Converts string data into double word [signed] data.	✓								Subsection 5.1.38
STR_TO_REAL(_E)	Converts string data into float (single precision) data.	✓								Subsection 5.1.39
STR_TO_TIME(_E)	Converts string data into time data.	✓								Subsection 5.1.40
BCD_TO_INT(_E)	Converts BCD data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.41
BCD_TO_DINT(_E)	Converts BCD data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.42
TIME_TO_BOOL(_E)	Converts time data into bit data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.43
TIME_TO_INT(_E)	Converts time data into word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.44
TIME_TO_DINT(_E)	Converts time data into double word [signed] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.45
TIME_TO_STR(_E)	Converts time data into string data.	✓								Subsection 5.1.46
TIME_TO_WORD(_E)	Converts time data into word [unsigned]/bit string [16-bit]data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.47
TIME_TO_DWORD (_E)	Converts time data into double word [unsigned]/bit string [32-bit] data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.1.48

*1. The function is provided in the FX3G Series Ver.1.10 or later.

2.2 Standard Functions Of One Numeric Variable

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
ABS(_E)	Obtains the absolute value.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.2.1

2.3 Standard Arithmetic Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
ADD_E	Adds data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.3.1
SUB_E	Subtracts data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.3.2
MUL_E	Multiplies data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.3.3
DIV_E	Divides data (, and outputs the quotient).	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.3.4
MOD(_E)	Divides data (, and outputs the remainder).	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.3.5
EXPT(_E)	Obtains the raised result.	✓								Subsection 5.3.6
MOVE(_E)	Transfers data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.3.7

2.4 Standard Bit Shift Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
SHL(_E)	Shifts bits leftward.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.4.1
SHR(_E)	Shifts bits rightward.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.4.2

2.5 Standard Bitwise Boolean Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
AND_E	Obtains the logical product.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.5.1
OR_E	Obtains the logical sum.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.5.2
XOR_E	Obtains the exclusive logical sum.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.5.3
NOT(_E)	Obtains the logical not.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.5.4

2.6 Standard Selection Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
SEL(_E)	Selects data in accordance with the input condition.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.6.1
MAXIMUM(_E)	Searches the maximum value.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.6.2
MINIMUM(_E)	Searches the minimum value.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.6.3
LIMITATION(_E)	Judges whether data is located within the range between the upper limit value and the lower limit value.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.6.4
MUX(_E)	Selects data, and outputs it.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.6.5

2.7 Standard Comparison Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
GT_E	Compares data with regard to "> (larger)".	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.7.1
GE_E	Compares data with regard to "≥ (larger or equal)".	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.7.2
EQ_E	Compares data with regard to "= (equal)".	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.7.3
LE_E	Compares data with regard to "≤ (smaller or equal)".	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.7.4
LT_E	Compares data with regard to "< (smaller)".	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.7.5
NE_E	Compares data with regard to "≠ (unequal)".	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.7.6

2.8 Standard Character String Functions

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
MID(_E)	Obtains a character string from a specified position.	✓								Subsection 5.8.1
CONCAT(_E)	Connects character strings.	✓								Subsection 5.8.2
INSERT(_E)	Inserts a character string.	✓								Subsection 5.8.3
DELETE(_E)	Deletes a character string.	✓								Subsection 5.8.4
REPLACE(_E)	Replaces a character string.	✓								Subsection 5.8.5
FIND(_E)	Searches a character string.	✓								Subsection 5.8.6

2.9 Functions Of Time Data Types

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
ADD_TIME(_E)	Adds time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.9.1
SUB_TIME(_E)	Subtracts time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.9.2
MUL_TIME(_E)	Multiplies time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.9.3
DIV_TIME(_E)	Divides time data.	✓	✓	✓	✓	✓	✓	✓	✓	Subsection 5.9.4

2.10 Standard Function Blocks

Function name	Function	Applicable PLC								Reference
		FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)	
R_TRIG(_E)	Detects the rising edge of a signal, and outputs pulse signal.	✓	✓	✓	✓	✓	✓	✓	✓	Section 6.1
F_TRIG(_E)	Detects the falling edge of a signal, and outputs pulse signal.	✓	✓	✓	✓	✓	✓	✓	✓	Section 6.2
CTU(_E)	Counts up the number of times of rising of a signal.	✓	✓	✓	✓	✓	✓			Section 6.3
CTD(_E)	Counts down the number of times of rising of a signal.	✓	✓	✓	✓	✓	✓			Section 6.4
CTUD(_E)	Counts up/down the number of times of rising of a signal.	✓	✓	✓	✓	✓	✓			Section 6.5
TP(_E)	Keeps ON a signal during specified time duration.	✓	✓	✓	✓	✓	✓			Section 6.6
TON(_E)	Keeps OFF a signal during specified time duration.	✓	✓	✓	✓	✓	✓			Section 6.7
TOF(_E)	Turns OFF the output signal at specified time after the input signal turned OFF.	✓	✓	✓	✓	✓	✓			Section 6.8
COUNTER_FB_M	Counter drive	✓	✓	✓	✓	✓	✓	✓	✓	Section 6.9
TIMER_10_FB_M	10ms timer drive	✓	✓	✓	✓	✓	✓	✓	✓	Section 6.10
TIMER_CONT_FB_M	Retentive timer drive	✓	✓	✓	✓		✓			Section 6.11
TIMER_100_M	100ms timer drive	✓	✓	✓	✓	✓	✓	✓	✓	Section 6.12

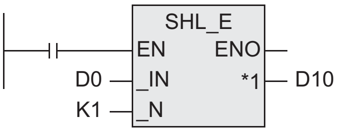
3. Function Construction

This chapter explains the construction of applied functions.

3.1 Applied Function Expression and Execution Type

Applied function and argument

- The name expressing the contents is given to each function.
  For example, the function name "SHL (bit shift left)" is given.
- Each function consists of arguments which indicate I/O data used in the function.



_IN (s) : An argument whose contents do not change even if the function is executed is called "source", and expressed in this symbol.

*1 (d) : An argument whose contents change when the function is executed is called "destination", and expressed in this symbol.

K1 (n) : Arguments not regarded as source or destination are expressed in "m", "n", etc.

Argument target devices

- The input variable (label or device) specifies the target.
- Bit device themselves such as X, Y, M and S may be handled.
- Bit devices may be combined in a way "KnX", "KnY", "KnM" and "KnS" to express numeric data.
  → **FX Structured Programming Manual (Device & Common)**
- Current value registers of data registers (D), timers (T) and counters (C) may be handled.
- When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects.
  Use labels when handling 32-bit data.
  You can specify 32-bit counters directly, however, because they have 32-bit length. Use global labels when specifying devices.
  When 32-bit data is handled, two consecutive 16-bit data registers D are combined.
  For example, when data register D0 is defined as an argument of a 32-bit instruction by a label, 32-bit data stored in D1 and D0 is handled. (D1 offers high-order 16 bits, and D0 offers low-order 16-bits.)
  When the current value register of a timer or counter is used as a general data register, it is handled in the same way.

3.2 Labels

Label types

Labels are classified into two types, global and local.

- Global labels can be used in program components and function blocks.
- Local labels can be used only in declared program blocks.

Label class

The label class indicates how each label can be used from which program component.
The table below shows label classes.

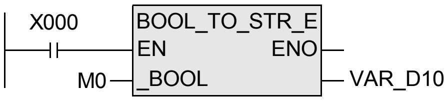
Class	Description	Applicable program component		
		Program block	Function	Function block
VAR_GLOBAL	Common label available in all program components	✓		✓
VAR_GLOBAL_CONSTANT	Common constant available in all program components	✓		✓
VAR	Label available within declared program components, and not available in any other program component	✓	✓	✓
VAR_CONSTANT	Constant available within declared program components, and not available in any other program component	✓	✓	✓
VAR_INPUT	Label which receives a value, and cannot be changed in program components		✓	✓
VAR_OUTPUT	Label output from a function block			✓
VAR_IN_OUT	Local label which receives a value, outputs it from a program component, and can be changed in program components			✓

Label definition

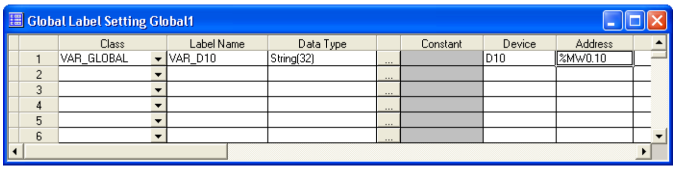
It is necessary to define a label to use the label.
An error will occur when a program in which labels are not defined is converted (compiled).

- When defining a global label, set the label name, class and data type, and assigns a device.
- When defining a local label, set the label name, class and data type.
  You do not have to specify devices for local labels. Assignment of devices is automatically executed during compiling.

In the example below, the label "VAR_D10" is set for the function "BOOL_TO_STR_E".



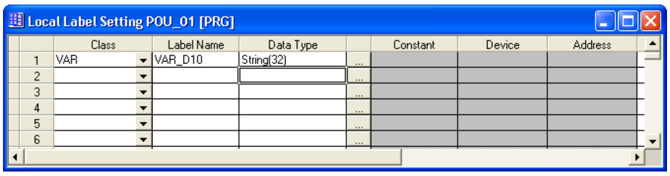
- When using "VAR_D10" as a global label
  Set the class, label name, data type and device (or address).



Global Label Setting Global1

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR_GLOBAL	VAR_D10	String(32)	...		D10	%MW0.10
2				...			
3				...			
4				...			
5				...			
6				...			

- When using "VAR_D10" as a local label
  Set the class, label name and data type.



Local Label Setting POU_01 [PRG]

	Class	Label Name	Data Type		Constant	Device	Address
1	VAR	VAR_D10	String(32)	...			
2				...			
3				...			
4				...			
5				...			
6				...			

Constant description method

The table below the description method required to set a constant to a label.

Constant type	Description method	Example
Bit	Input "TRUE" or "FALSE". Or input "0" or "1".	TRUE, FALSE
Binary number	Add "2#" before a binary number.	2#0010, 2#01101010
Octal number	Add "8#" before an octal number.	8#0, 8#337
Decimal number	Input a decimal number directly. Or add "K" before a decimal number.	123, K123
Hexadecimal number	Add "16#" or "H" before a hexadecimal number.	16#FF, HFF
Real number	Input a real number directly. Or add "E" before a real number.	2.34, E2.34
Character string	Surround a character string with single quotations (') or double quotations (").	'ABC', "ABC"

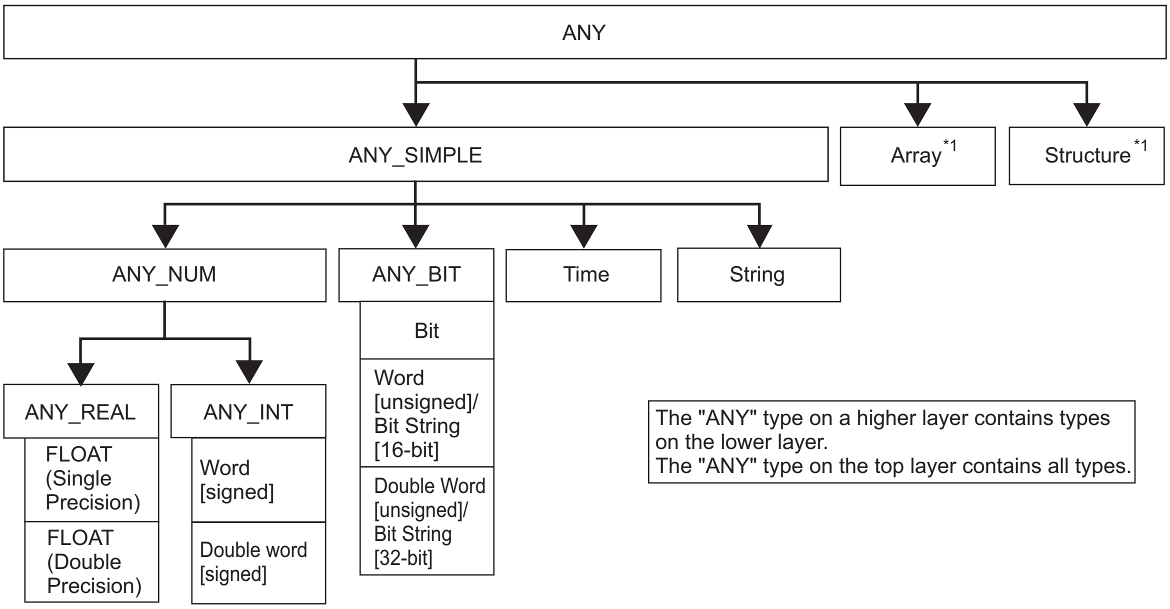
Data type

The label data type is basic or universal.

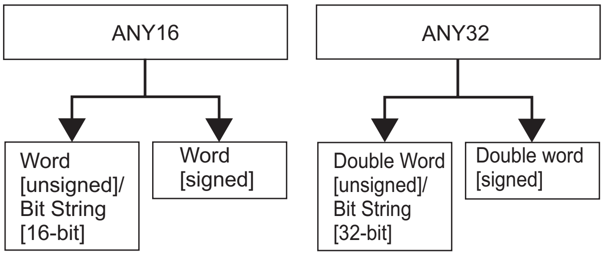
- The table below shows a list of basic data types.

Data type	Description	Value range	Bit length
Bit	Boolean data	0(FALSE), 1(TRUE)	1 bit
Word [signed]	Integer	-32768 to 32767	16 bits
Double Word [signed]	Double precision integer	-2147483648 to 2147483647	32 bits
Word [unsigned]/Bit String [16-bit]	16-bit data	0 to 65535	16 bits
Double Word [unsigned]/Bit String [32-bit]	32-bit data	0 to 4294967295	32 bits
FLOAT (Single Precision)	Real number	E $\pm1.175495^{-38}$ to E $\pm3.402823^{+38}$ (Number of significant figures: 6)	32 bits
String	Character string	(50 characters maximum)	Variable
Time	Time value	T#-24d-0h31m23s648.00ms to T#24d20h31m23s647.00ms	32 bits

- The universal data type indicates data type of a label which combines several basic data types. The data type name begins with "ANY".



The "ANY" type on a higher layer contains types on the lower layer.
The "ANY" type on the top layer contains all types.


ANY16
Word [unsigned]/ Bit String [16-bit]
Word [signed]
ANY32
Double Word [unsigned]/ Bit String [32-bit]
Double word [signed]

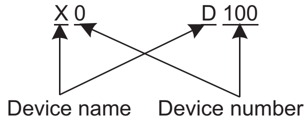
*1 Refer to the following manual for details.
→ Q/FX Structured Programming Manual (Fundamentals)

3.3 Device and Address

Devices can be described in two methods, device method and address method.

Device method

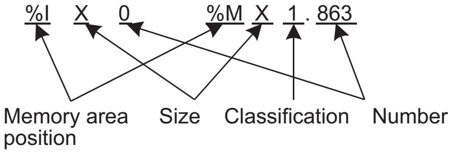
In this method, a device is described using the device name and device number.



Address method

This method is defined in IEC61131-3, and used as shown in the table below.

Head	1st character: Position		2nd character: Size		3rd and later characters: Classification	Number
%	I	Input	(Omitted)	Bit	This number is provided for detailed classification. Period (.) is used to delimit the subsequent "Number". The characters for classification may be omitted.	This decimal number corresponds to the device number.
	Q	Output	X	Bit		
	M	Internal	W	Word (16 bits)		
			D	Double word (32 bits)		
			L	Long Word (64 bits)		



- Memory area position
  The memory area position in which data is assigned is classified into "input", "output" or "internal".
  X(X Device method) : I(Input)
  Y(Y Device method) : Q(Output)
  Any other device : M(Internal)
- Size
  The principle of the description method corresponding to the device method (MELSEC description method) is as follows:
  Bit device : X(Bit)
  Word device : W(Word (16 bits)), D(Double word (32 bits))
- Classification
  The 3rd and later characters indicate the device type which cannot be specified only by the position and size explained above.
  The classification is not required for devices "X" and "Y".

Refer to the following for the device description method:

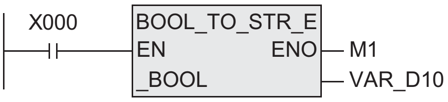
→ **7.3 Appendix A**

3.4 EN and ENO

Execution of an instruction can be controlled when the instruction contains "EN" in its name.

- "EN" inputs the instruction execution condition.
- "ENO" outputs the instruction execution status.
- The table below shows the "ENO" status corresponding to the "EN" status and the operation result.

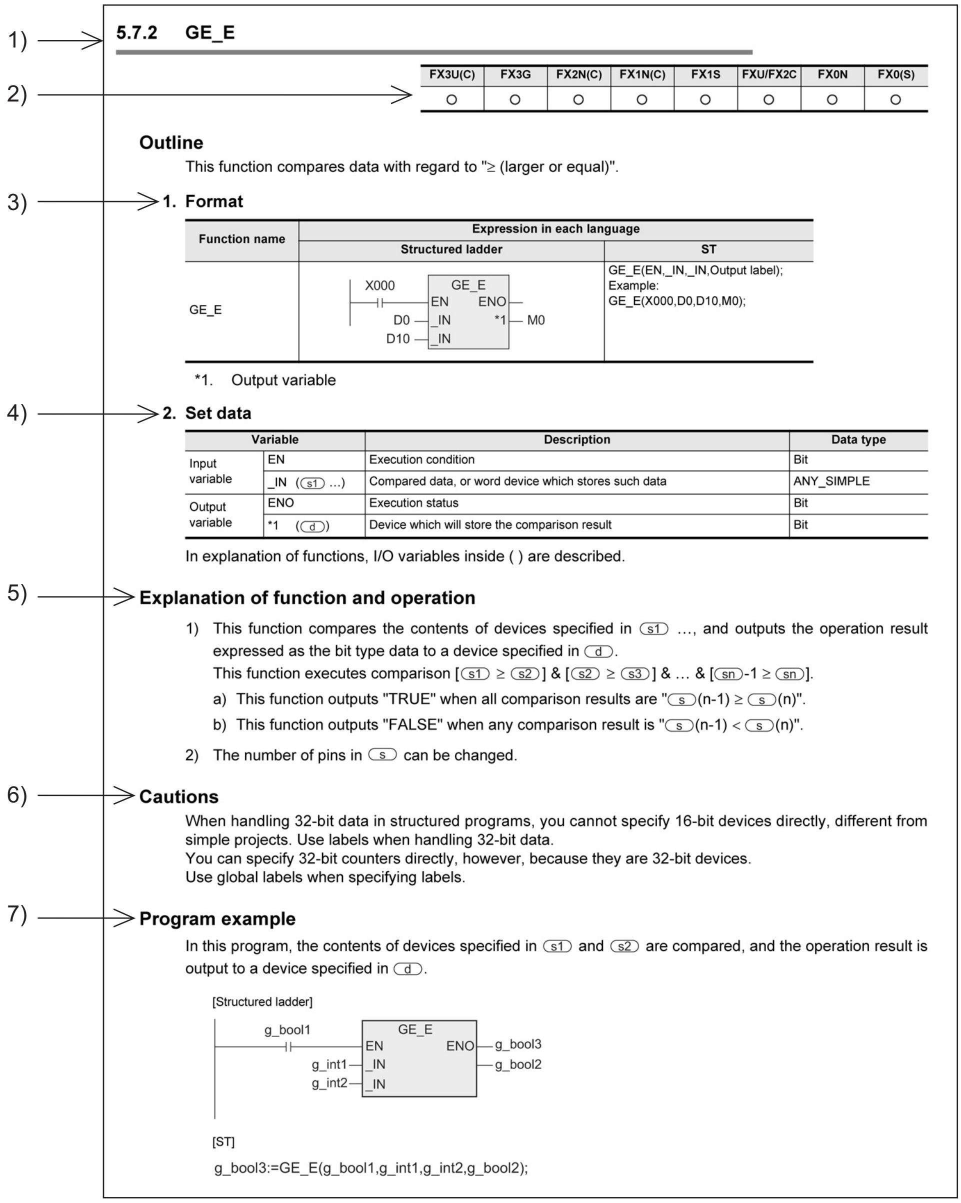
EN	ENO	Operation result
TRUE(Executes operation.)	TRUE(Operation error did not occur.)	Operation output value
	FALSE(Operation error occurred.)	Indefinite value
FALSE(Stops operation.)	FALSE	Indefinite value



In the above example, the function "BOOL_TO_STR_E" is executed only when X000 is "TRUE".
When the function is executed normally, "TRUE" is output to M1.

4. How to Read Explanation of Functions

Function explanation pages have the following configuration.



1) 5.7.2 GE_E

2)

FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≥ (larger or equal)".

3) **1. Format**

Function name	Expression in each language	
	Structured ladder	ST

GE_E	X000
	GE_E
	EN
	ENO
	D0
	_IN
	*1
	M0
	D10
	_IN
		GE_E(EN,_IN,_IN,Output label);
		Example:
		GE_E(X000,D0,D10,M0);

*1. Output variable

4) **2. Set data**

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

5) **Explanation of function and operation**

1) This function compares the contents of devices specified in s1 ..., and outputs the operation result expressed as the bit type data to a device specified in d.
This function executes comparison [s1 ≥ s2] & [s2 ≥ s3] & ... & [sn-1 ≥ sn].
a) This function outputs "TRUE" when all comparison results are "s(n-1) ≥ s(n)".
b) This function outputs "FALSE" when any comparison result is "s(n-1) < s(n)".

2) The number of pins in s can be changed.

6) **Cautions**

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

7) **Program example**

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder]

[ST]

```
g_bool3:=GE_E(g_bool1,g_int1,g_int2,g_bool2);
```

* The above page is prepared for explanation, and is different from the actual page.

1) Indicates the chapter/section/subsection number and instruction name.

2) Indicates PLCs which support the function.

Item	Description
○	The PLC Series supports the function from its first product.
△	The supporting status varies on the version. Applicable versions are explained in "Cautions".
×	The PLC Series does not support the function.

3) Indicates the expression of each function.

Item	Description
Structured ladder	Indicates the instruction expression in the structured ladder language.
ST	Indicates the instruction expression in the ST language.

4) Indicates the input variable name and output variable name of the function as well as the contents and data type of each variable.
Refer to the following for detailed data types:
→ **Q/FX Structured Programming Manual (Fundamentals)**

5) Explanation of function and operation
The function executed by this function is explained.
In explanation, the structured ladder language is used as the representative.

6) Cautions
Cautions on using the function are described.

7) Program example
Program examples are explained in each language.

5. Applied Functions

This chapter explains the operation outline of each applied function, symbols, I/O data type, equivalent circuit in sequence instructions, target models, cautions and program examples.
Refer to the following manual for variables, operators, data types and program languages:
→ **Q/FX Structured Programming Manual (Fundamentals)**

5.1 Type Conversion Functions

5.1.1 BOOL_TO_INT(_E)

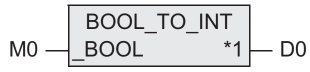
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

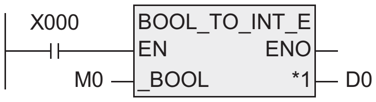
Outline

This function coverts bit data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BOOL_TO_INT		BOOL_TO_INT(_BOOL);
		Example:
		D0:=
		BOOL_TO_INT(M0);

BOOL_TO_INT_E	 X000 BOOL_TO_INT_E EN ENO M0 _BOOL *1 D0	BOOL_TO_INT_E(EN,_BOOL, Output label Example: BOOL_TO_INT_E(X000,M0, D0);

*1. Output variable

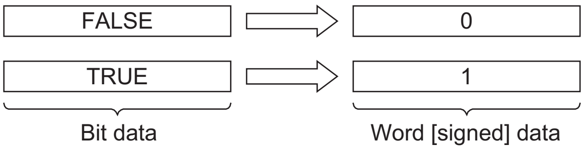
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data stored in a device specified in s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.
When the input value is "FALSE", this function outputs "0" as the word [signed] data value.
When the input value is "TRUE", this function outputs "1" as the word [signed] data value.



Cautions

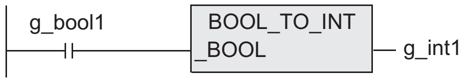
Use the function having "_E" in its name to connect a bus.

Program example

In this program, bit data stored in a device specified in (s) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_INT)

[Structured ladder]

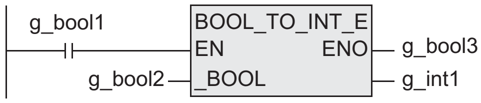


[ST]

```
g_int1 := BOOL_TO_INT(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := BOOL_TO_INT_E(g_bool1, g_bool2, g_int1);
```

5.1.2 BOOL_TO_DINT(_E)

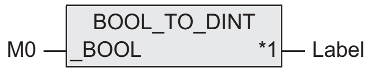
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

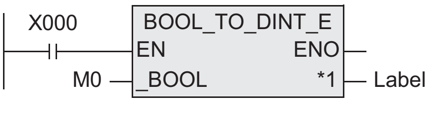
Outline

This function converts bit data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BOOL_TO_DINT		BOOL_TO_DINT(_BOOL);
		Example:
		Label:=
		BOOL_TO_DINT(M0);

BOOL_TO_DINT_E	 X000
	BOOL_TO_DINT_E
	EN
	ENO
	M0
	_BOOL
	*1
	Label
		BOOL_TO_DINT_E(EN, _BOOL, Output label);
		Example:
		BOOL_TO_DINT_E(X000,M0, Label);

*1. Output variable

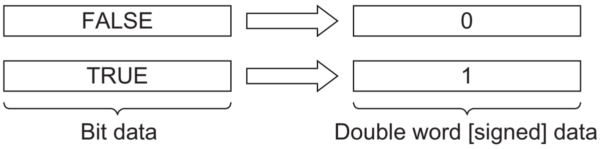
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data stored in a device specified in s into double word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

1) Use the function having "_E" in its name to connect a bus.

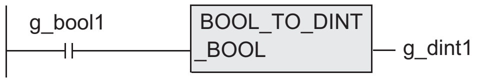
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, bit data stored in a device specified in (s) is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_DINT)

[Structured ladder]

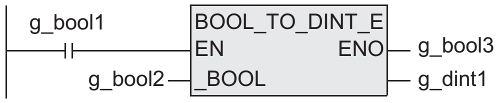


[ST]

```
g_dint1 := BOOL_TO_DINT(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := BOOL_TO_DINT_E(g_bool1, g_bool2, g_dint1);
```

5.1.3 BOOL_TO_STR(_E)

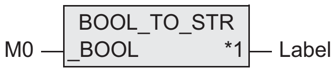
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

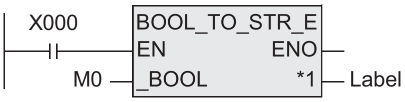
Outline

This function converts bit data into string data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BOOL_TO_STR		BOOL_TO_STR(_BOOL);
		Example:
		Label:=
		BOOL_TO_STR(M0);

BOOL_TO_STR_E	 X000 M0 BOOL_TO_STR_E EN ENO _BOOL *1 Label	BOOL_TO_STR_E(EN, _BOOL, Output label); Example: BOOL_TO_STR_E(X000,M0, Label);

*1. Output variable

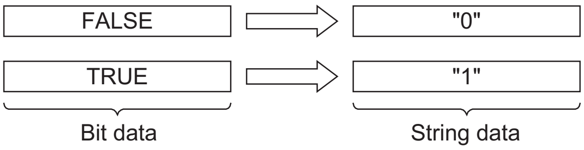
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data input to a deice specified in s into string data, and outputs the data obtained by conversion to a device specified in d.



Cautions

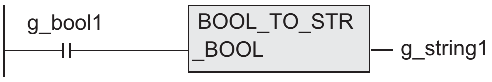
1) Use the function having "_E" in its name to connect a bus.
2) When handling string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data. Use global labels when specifying labels.

Program example

In this program, bit data stored in a deice specified in (s) is converted into string data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_STR)

[Structured ladder]

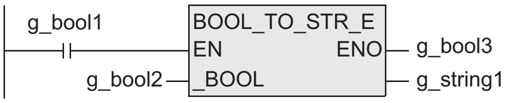


[ST]

```
g_string1 := BOOL_TO_STR(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_STR_E)

[Structured ladder]



[ST]

```
g_bool3 := BOOL_TO_STR_E(g_bool1, g_bool2, g_string1);
```

5.1.4 BOOL_TO_WORD(_E)

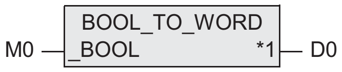
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

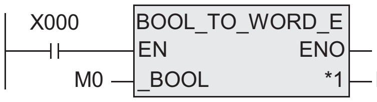
Outline

This function converts bit data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BOOL_TO_WORD		BOOL_TO_WORD(_BOOL);
		Example:
		D0:=
		BOOL_TO_WORD(M0);

BOOL_TO_WORD_E	 X000 M0 BOOL_TO_WORD_E EN ENO _BOOL *1 D0	BOOL_TO_WORD_E(EN, _BOOL, Output label); Example: BOOL_TO_WORD_E(X000, M0,D0);

*1. Output variable

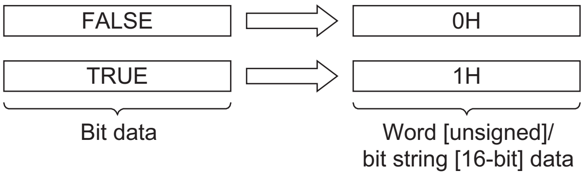
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data stored in a device specified in s into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.
When the input value is "FALSE", this function outputs "0H" as the word [unsigned]/bit string [16-bit] data value.
When the input value is "TRUE", this function outputs "1H" as the word [unsigned]/bit string [16-bit] data value.



Cautions

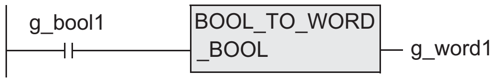
Use the function having "_E" in its name to connect a bus.

Program example

In this program, bit data stored in a device specified in s is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(BOOL_TO_WORD)

[Structured ladder]

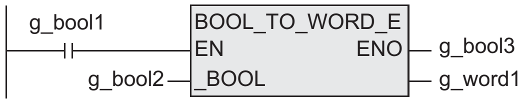


[ST]

```
g_word1 := BOOL_TO_WORD(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_WORD_E)

[Structured ladder]



[ST]

```
g_bool3 := BOOL_TO_WORD_E(g_bool1, g_bool2, g_word1);
```

5.1.5 BOOL_TO_DWORD(_E)

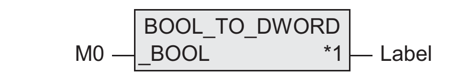
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

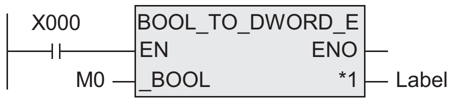
Outline

This function converts bit data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BOOL_TO_DWORD		BOOL_TO_DWORD(_BOOL);
		Example:
		Label:=
		BOOL_TO_DWORD(M0);

BOOL_TO_DWORD_E	 X000 BOOL_TO_DWORD_E EN ENO M0 _BOOL *1 Label	BOOL_TO_DWORD_E(EN, _BOOL, Output label); Example: BOOL_TO_DWORD_E(X000, M0, Label);

*1. Output variable

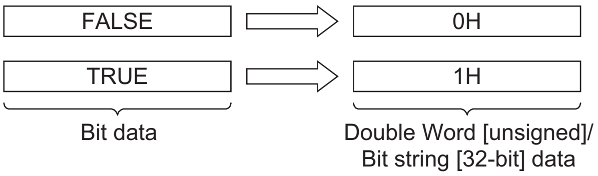
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data stored in a device specified in s into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in d.
When the input value is "FALSE", this function outputs "0H" as the double word [unsigned]/bit string [32-bit]data value.
When the input value is "TRUE", this function outputs "1H" as the double word [unsigned]/bit string [32-bit] data value.



Cautions

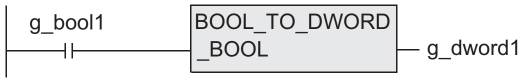
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, bit data stored in a device specified in (s) is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_DWORD)

[Structured ladder]

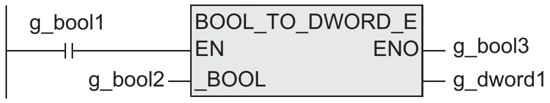


[ST]

```
g_dword1 := BOOL_TO_DWORD(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_DWORD_E)

[Structured ladder]



[ST]

```
g_bool3 := BOOL_TO_DWORD_E(g_bool1, g_bool2, g_dword1);
```

5.1.6 BOOL_TO_TIME(_E)

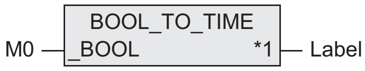
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

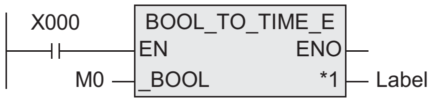
Outline

This function converts bit data into time data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BOOL_TO_TIME		BOOL_TO_TIME(_BOOL);
		Example:
		Label:=
		BOOL_TO_TIME(M0);

BOOL_TO_TIME_E	 X000 BOOL_TO_TIME_E EN ENO M0 _BOOL *1 Label	BOOL_TO_TIME_E(EN,_BOOL, Output label); Example: BOOL_TO_TIME_E(X000,M0, Label);

*1. Output variable

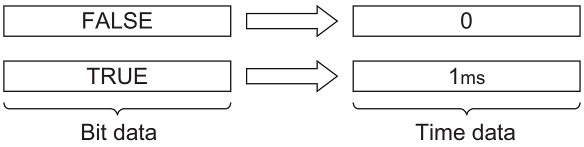
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BOOL (s)	Conversion source bit data	Bit
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts bit data stored in a device specified in (s) into time data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

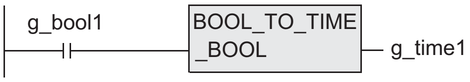
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, bit data stored in a device specified in (s) is converted into time data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(BOOL_TO_TIME)

[Structured ladder]

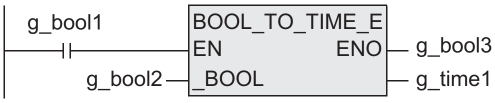


[ST]

```
g_time1 := BOOL_TO_TIME(g_bool1);
```

2) Function with EN/ENO(BOOL_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := BOOL_TO_TIME_E(g_bool1, g_bool2, g_time1);
```

5.1.7 INT_TO_DINT(_E)

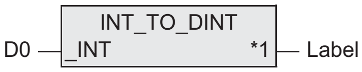
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

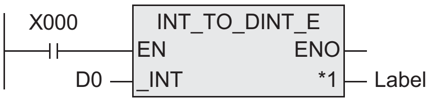
Outline

This function converts word [signed] data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_DINT		INT_TO_DINT(_INT);
		Example:
		Label:=
		INT_TO_DINT(D0);

INT_TO_DINT_E	 X000 INT_TO_DINT_E EN ENO D0 _INT *1 Label	INT_TO_DINT_E(EN,_INT, Output label); Example: INT_TO_DINT_E(X000,D0, Label);

*1. Output variable

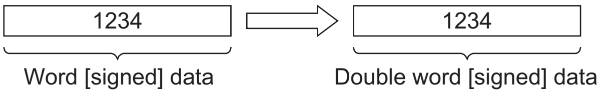
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in s into double word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

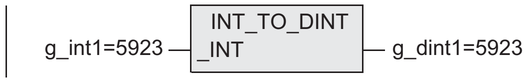
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, word [signed] data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_DINT)

[Structured ladder]

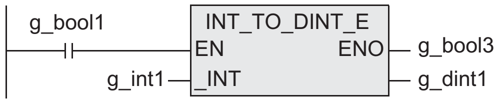


[ST]

```
g_dint1 := INT_TO_DINT(g_int1);
```

2) Function with EN/ENO(INT_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_DINT_E(g_bool1, g_int1, g_dint1);
```

5.1.8 DINT_TO_INT(_E)

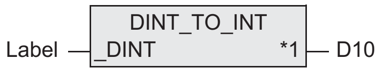
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

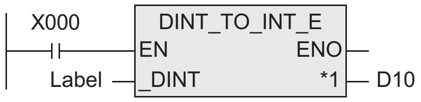
Outline

This function converts double word [signed] data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_INT		DINT_TO_INT(_DINT);
		Example:
		D10:=
		DINT_TO_INT(Label);

DINT_TO_INT_E	 X000 DINT_TO_INT_E EN ENO Label _DINT *1 D10	DINT_TO_INT_E(EN,_DINT, Output label); Example: DINT_TO_INT_E(X000, Label, D10);

*1. Output variable

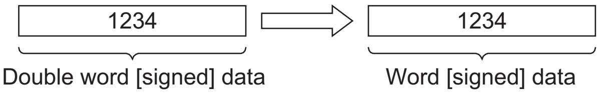
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

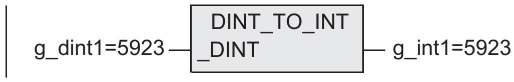
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, double word [signed] data stored in a device specified in s is converted into word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_INT)

[Structured ladder]

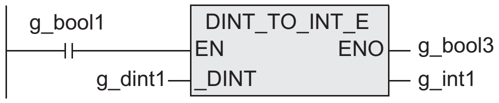


[ST]

```
g_int1 := DINT_TO_INT(g_dint1);
```

2) Function with EN/ENO(DINT_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_INT_E(g_bool1, g_dint1, g_int1);
```

5.1.9 INT_TO_BOOL(_E)

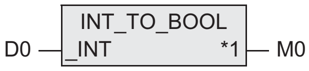
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

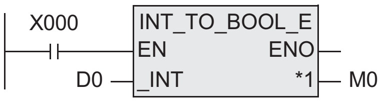
Outline

This function converts word [signed] data into bit data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_BOOL		INT_TO_BOOL(_INT);
		Example:
		M0:=
		INT_TO_BOOL(D0);

INT_TO_BOOL_E	 X000
	INT_TO_BOOL_E
	EN
	ENO
	D0
	_INT
	*1
	M0
		INT_TO_BOOL_E(EN,_INT,
		Output label);
		Example:
		INT_TO_BOOL_E(X000,D0,M0);

*1. Output variable

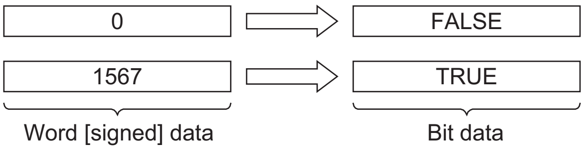
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).
When the input value is "0", this function outputs "FALSE".
When the input value is any value other than "0", this function outputs "TRUE".



Cautions

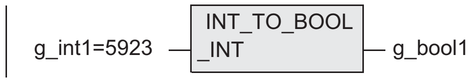
Use the function having "_E" in its name to connect a bus.

Program example

In this program, word [signed] data stored in a device specified in (s) is converted into bit data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(INT_TO_BOOL)

[Structured ladder]

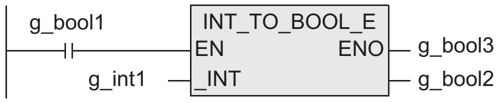


[ST]

```
g_bool1 := INT_TO_BOOL(g_int1);
```

2) Function with EN/ENO(INT_TO_BOOL_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_BOOL_E(g_bool1, g_int1, g_bool2);
```

5.1.10 DINT_TO_BOOL(_E)

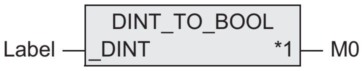
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

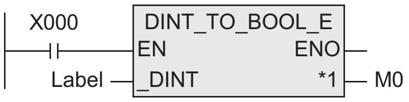
Outline

This function converts double word [signed] data into bit data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_BOOL		DINT_TO_BOOL(_DINT);
		Example:
		M0:=
		DINT_TO_BOOL(Label);

DINT_TO_BOOL_E	 X000
	DINT_TO_BOOL_E
	EN
	ENO
	Label
	_DINT
	*1
	M0
		DINT_TO_BOOL_E(EN,_DINT, Output label);
		Example:
		DINT_TO_BOOL_E(X000, Label, M0);

*1. Output variable

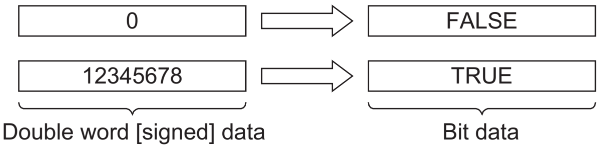
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in s into bit data, and outputs the data obtained by conversion to a device specified in d.
When the input value is "0", this function outputs "FALSE".
When the input value is any value other than "0", this function outputs "TRUE".



Cautions

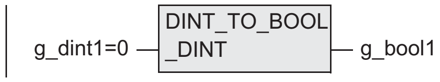
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [signed] data stored in a device specified in s is converted into bit data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_BOOL)

[Structured ladder]

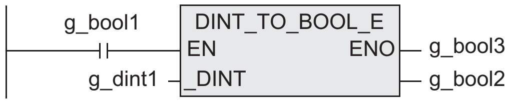


[ST]

```
g_bool1 := DINT_TO_BOOL(g_dint1);
```

2) Function with EN/ENO(DINT_TO_BOOL_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_BOOL_E(g_bool1, g_dint1, g_bool2);
```

5.1.11 INT_TO_REAL(_E)

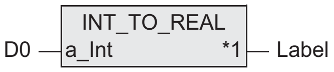
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	×	×	×	×	×

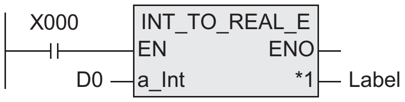
Outline

This function converts word [signed] data into float (single precision) data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_REAL		INT_TO_REAL(a_Int);
		Example:
		Label:=
		INT_TO_REAL(D0);

INT_TO_REAL_E	 X000
	INT_TO_REAL_E
	EN
	ENO
	D0
	a_Int
	*1
	Label
		INT_TO_REAL_E(EN,a_Int,
		Output label);
		Example:
		INT_TO_REAL_E(X000,D0,Label);

*1. Output variable

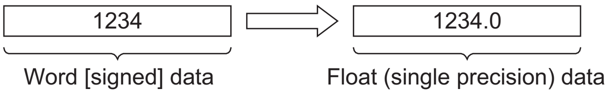
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_Int (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Float (single precision) data after conversion	FLOAT (Single Precision)

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in (s) into float (single precision) data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

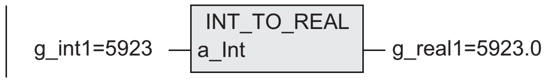
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.

Program example

In this program, word [signed] data stored in a device specified in (s) is converted into float (single precision) data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(INT_TO_REAL)

[Structured ladder]

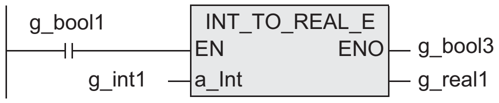


[ST]

```
g_real1 := INT_TO_REAL(g_int1);
```

2) Function with EN/ENO(INT_TO_REAL_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_REAL_E(g_bool1, g_int1, g_real1);
```

5.1.12 DINT_TO_REAL(_E)

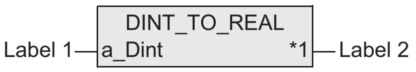
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	×	×	×	×	×

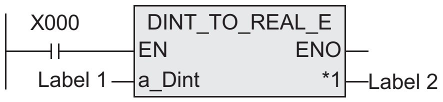
Outline

This function converts double word [signed] data into float (single precision) data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_REAL		DINT_TO_REAL(a_Dint);
		Example:
		Label 2:=
		DINT_TO_REAL(Label 1);

DINT_TO_REAL_E	 X000 DINT_TO_REAL_E EN ENO Label 1 a_Dint *1 Label 2	DINT_TO_REAL_E(EN,a_Dint, Output label); Example: DINT_TO_REAL_E(X000, Label 1, Label 2);

*1. Output variable

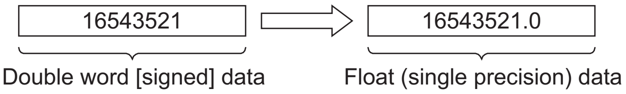
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_Dint (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Float (single precision) data after conversion	FLOAT (Single Precision)

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in (s) into float (single precision) data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

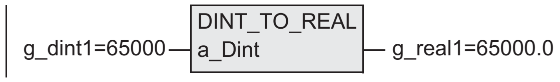
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.

Program example

In this program, double word [signed] data stored in a device specified in s is converted into float (single precision) data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_REAL)

[Structured ladder]

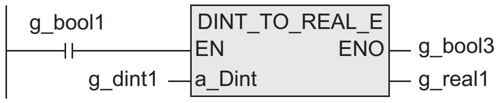


[ST]

```
g_real1 := DINT_TO_REAL(g_dint1);
```

2) Function with EN/ENO(DINT_TO_REAL_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_REAL_E(g_bool1, g_dint1, g_real1);
```

5.1.13 INT_TO_STR(_E)

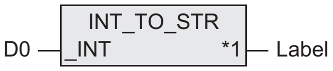
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

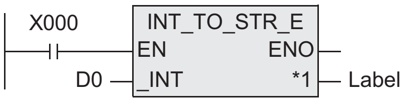
Outline

This function converts word [signed] data into string data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_STR		INT_TO_STR(_INT);
		Example:
		Label:=
		INT_TO_STR(D0);

INT_TO_STR_E	 X000
	INT_TO_STR_E
	EN
	ENO
	D0
	_INT
	*1
	Label
		INT_TO_STR_E(EN,_INT,
		Output label);
		Example:
		INT_TO_STR_E(X000, D0, Label);

*1. Output variable

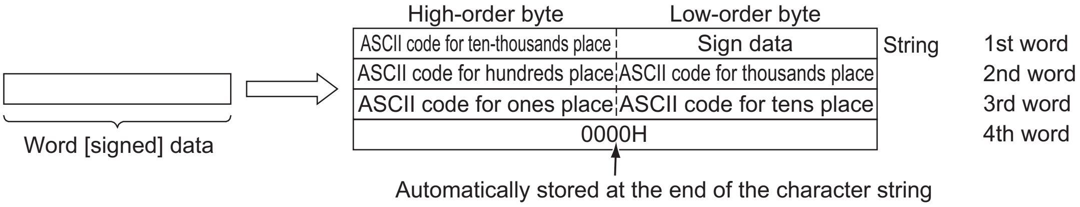
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts word [signed] data stored in a device specified in (s) into string data, and outputs the data obtained by conversion to a device specified in (d).

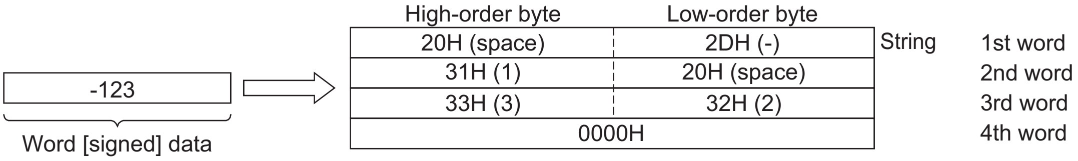


	High-order byte	Low-order byte		
	ASCII code for ten-thousands place	Sign data	String	1st word
	ASCII code for hundreds place	ASCII code for thousands place		2nd word
	ASCII code for ones place	ASCII code for tens place		3rd word
	0000H			4th word

Automatically stored at the end of the character string

2) In "Sign data", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

3) "20H (space)" is stored in high-order digits when the number of significant figures is small.
Example: When "-123" is input



	High-order byte	Low-order byte		
	20H (space)	2DH (-)	String	1st word
	31H (1)	20H (space)		2nd word
	33H (3)	32H (2)		3rd word
	0000H			4th word

4) "00H" is automatically stored at the end (4th word) of the character string.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following case. The error flag M8067 turns ON, and D8067 stores the error code.

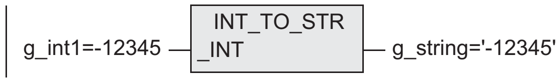
1) When the number of points occupied by the string data storage destination (device specified in d) exceeds the range of the corresponding device
(Error code: K6706)

Program example

In this program, word [signed] data stored in a device specified in s is converted into string data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_STR)

[Structured ladder]

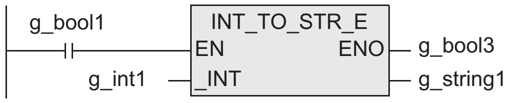


[ST]

```
g_string1 := INT_TO_STR(g_int1);
```

2) Function with EN/ENO(INT_TO_STR_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_STR_E(g_bool1, g_int1, g_string1);
```

5.1.14 DINT_TO_STR(_E)

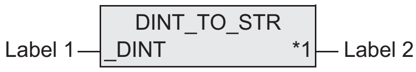
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

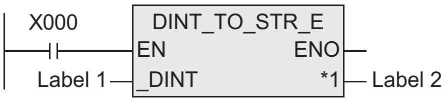
Outline

This function converts double word [signed] data into string data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_STR		DINT_TO_STR(_DINT);
		Example:
		Label 2:=
		DINT_TO_STR(Label 1);

DINT_TO_STR_E	 X000 DINT_TO_STR_E EN ENO Label 1 _DINT *1 Label 2	DINT_TO_STR_E(EN,_DINT, Output label); Example: DINT_TO_STR_E(X000, Label 1, Label 2);

*1. Output variable

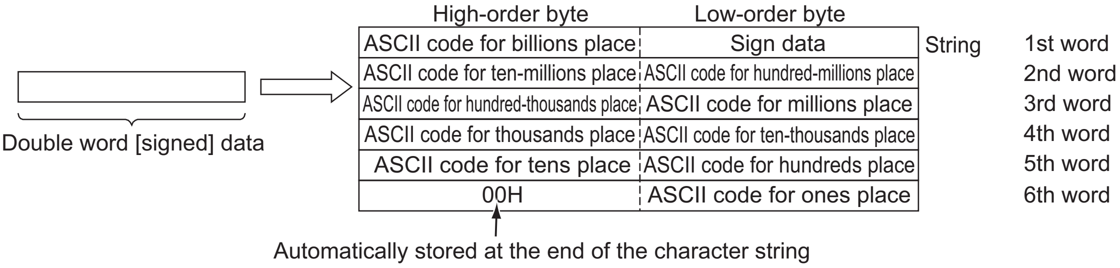
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts double word [signed] data stored in a device specified in (s) into string data, and outputs the data obtained by conversion to a device specified in (d).



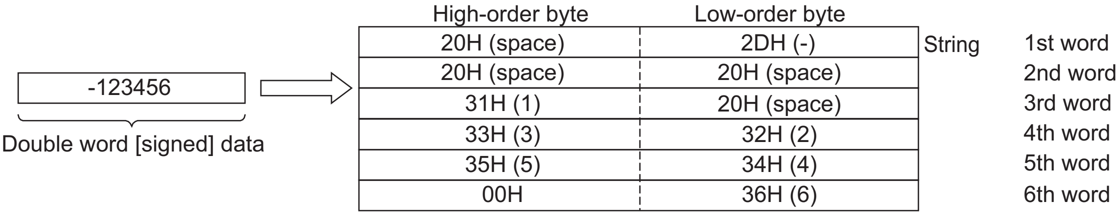
Double word [signed] data

	High-order byte	Low-order byte		
String	ASCII code for billions place	Sign data	1st word
	ASCII code for ten-millions place	ASCII code for hundred-millions place	2nd word
	ASCII code for hundred-thousands place	ASCII code for millions place	3rd word
	ASCII code for thousands place	ASCII code for ten-thousands place	4th word
	ASCII code for tens place	ASCII code for hundreds place	5th word
	00H	ASCII code for ones place	6th word

Automatically stored at the end of the character string

2) In "Sign data", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

3) "20H (space)" is stored in high-order digits when the number of significant figures is small.
Example: When "-123456" is input



-123456

Double word [signed] data

	High-order byte	Low-order byte	
String	20H (space)	2DH (-)	1st word
	20H (space)	20H (space)	2nd word
	31H (1)	20H (space)	3rd word
	33H (3)	32H (2)	4th word
	35H (5)	34H (4)	5th word
	00H	36H (6)	6th word

4) "00H" is automatically stored at the end (high-order byte of the 6th word) of the character string.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Error

An operation error occurs in the following case. The error flag M8067 turns ON, and D8067 stores the error code.

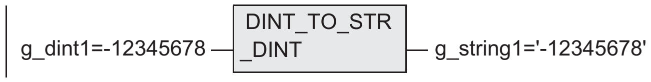
1) When the number of points occupied by the string data storage destination (device specified in d) exceeds the range of the corresponding device
   (Error code: K6706)

Program example

In this program, double word [signed] data stored in a device specified in s is converted into string data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_STR)

[Structured ladder]

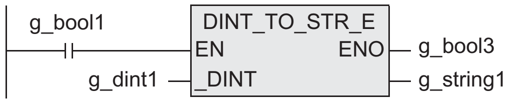


[ST]

```
g_string1 := DINT_TO_STR(g_dint1);
```

2) Function with EN/ENO(DINT_TO_STR_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_STR_E(g_bool1, g_dint1, g_string1);
```

5.1.15 INT_TO_WORD(_E)

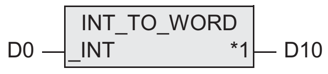
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

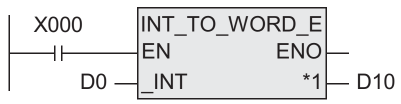
Outline

This function converts word [signed] data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_WORD		INT_TO_WORD(_INT);
		Example:
		D10:=
		INT_TO_WORD(D0);

INT_TO_WORD_E	 X000
	INT_TO_WORD_E
	EN
	ENO
	D0
	_INT
	*1
	D10
		INT_TO_WORD_E(EN,_INT,
		Output label);
		Example:
		INT_TO_WORD_E(X000,D0,D10);

*1. Output variable

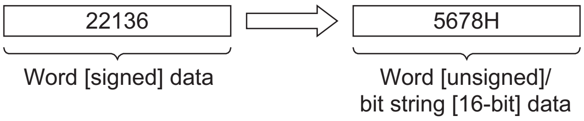
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/Bit String [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in (s) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

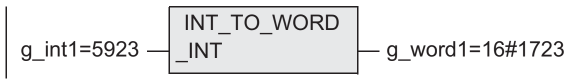
Use the function having "_E" in its name to connect a bus.

Program example

In this program, word [signed] data stored in a device specified in s is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_WORD)

[Structured ladder]

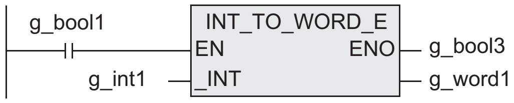


[ST]

```
g_word1 := INT_TO_WORD(g_int1);
```

2) Function with EN/ENO(INT_TO_WORD_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_WORD_E(g_bool1, g_int1, g_word1);
```

5.1.16 DINT_TO_WORD(_E)

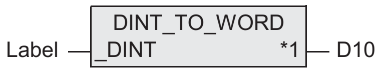
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

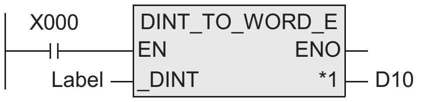
Outline

This function converts double word [signed] data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_WORD		DINT_TO_WORD(_DINT);
		Example:
		D10:=
		DINT_TO_WORD(Label);

DINT_TO_WORD_E	 X000
	Label
	DINT_TO_WORD_E
	EN
	ENO
	_DINT
	*1
	D10
		DINT_TO_WORD_E(EN,_DINT, Output label);
		Example:
		DINT_TO_WORD_E(X000, Label, D10);

*1. Output variable

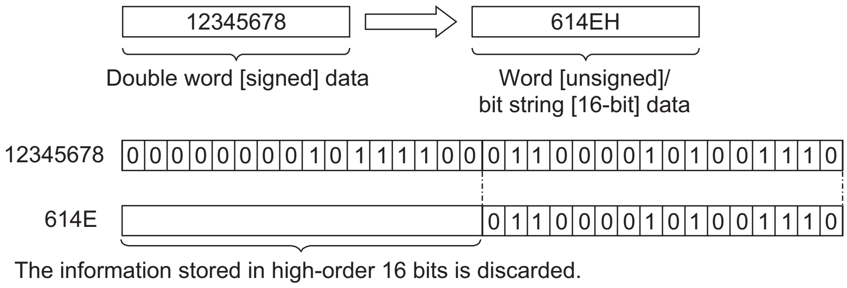
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in s into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

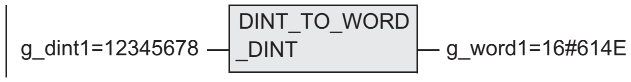
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [signed] data stored in a device specified in s is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_WORD)

[Structured ladder]

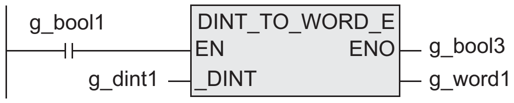


[ST]

g_word1 := DINT_TO_WORD(g_dint1);

2) Function with EN/ENO(DINT_TO_WORD_E)

[Structured ladder]



[ST]

g_bool3 := DINT_TO_WORD_E(g_bool1, g_dint1, g_word1);

5.1.17 INT_TO_DWORD(_E)

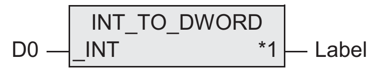
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

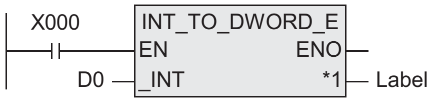
Outline

This function converts word [signed] data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_DWORD		INT_TO_DWORD(_INT);
		Example:
		Label:=
		INT_TO_DWORD(D0);

INT_TO_DWORD_E	 X000 INT_TO_DWORD_E EN ENO D0 _INT *1 Label	INT_TO_DWORD_E(EN,_INT, Output label); Example: INT_TO_DWORD_E(X000,D0, Label);

*1. Output variable

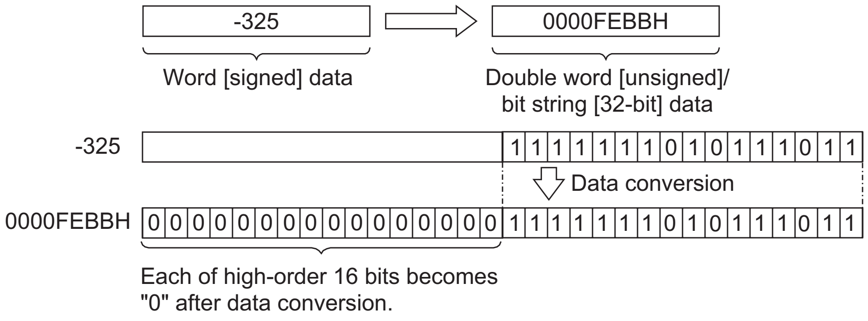
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in s into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

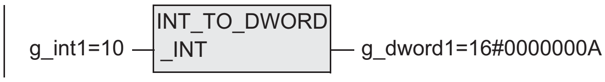
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.

Program example

In this program, word [signed] data stored in a device specified in s is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_DWORD)

[Structured ladder]

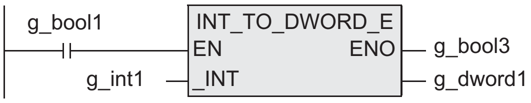


[ST]

```
g_dword1 := INT_TO_DWORD(g_int1);
```

2) Function with EN/ENO(INT_TO_DWORD_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_DWORD_E(g_bool1, g_int1, g_dword1);
```

5.1.18 DINT_TO_DWORD(_E)

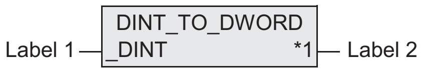
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

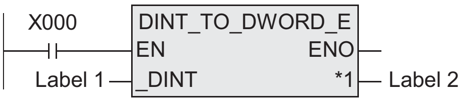
Outline

This function converts double word [signed] data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_DWORD		DINT_TO_DWORD(_DINT);
		Example:
		Label 2:=
		DINT_TO_DWORD(Label 1);

DINT_TO_DWORD_E	 X000
	DINT_TO_DWORD_E
	EN
	ENO
	_DINT
	*1
	Label 1
	Label 2
		DINT_TO_DWORD_E(EN,_DINT, Output label);
		Example:
		DINT_TO_DWORD_E(X000, Label 1, Label 2);

*1. Output variable

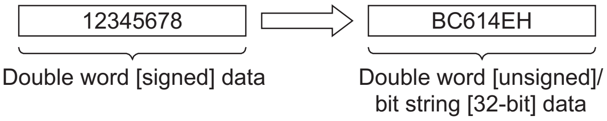
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (ⓢ)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in ⓢ into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in ⓓ.



Cautions

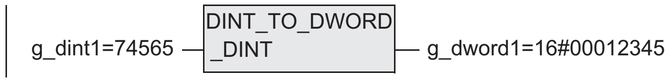
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [signed] data stored in a device specified in s is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_DWORD)

[Structured ladder]

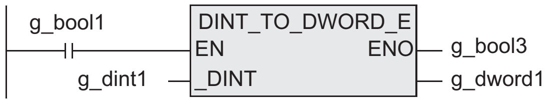


[ST]

```
g_dword1 := DINT_TO_DWORD(g_dint1);
```

2) Function with EN/ENO(DINT_TO_DWORD_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_DWORD_E(g_bool1, g_dint1, g_dword1);
```

5.1.19 INT_TO_BCD(_E)

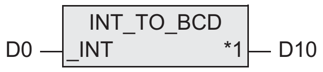
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

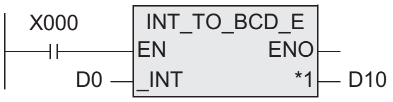
Outline

This function converts word [signed] data into BCD data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_BCD		INT_TO_BCD(_INT);
		Example:
		D10:=
		INT_TO_BCD(D0);

INT_TO_BCD_E	 X000
	INT_TO_BCD_E
	EN
	ENO
	D0
	_INT
	*1
	D10
		INT_TO_BCD_E(EN,_INT,
		Output label);
		Example:
		INT_TO_BCD_E(X000,D0,D10);

*1. Output variable

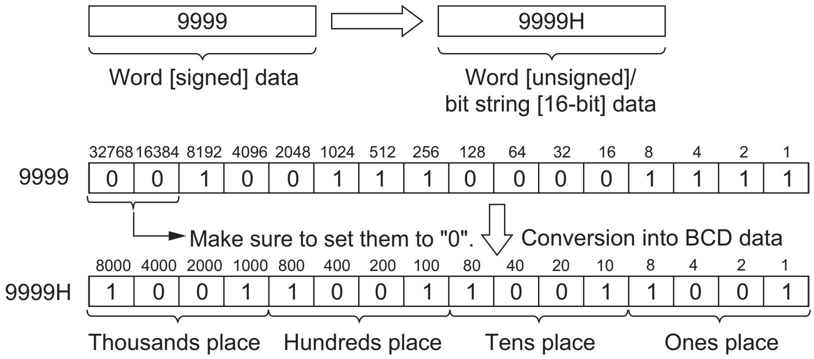
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	BCD data after conversion	Word [unsigned]/ Bit String [16-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in (s) into BCD data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

Use the function having "_E" in its name to connect a bus.

Error

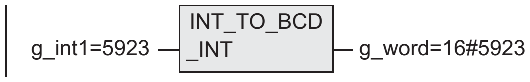
An operation error occurs when the value stored in a device specified in (s) is outside the range from "0" to "9,999".

Program example

In this program, word [signed] data stored in a device specified in s is converted into BCD data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_BCD)

[Structured ladder]

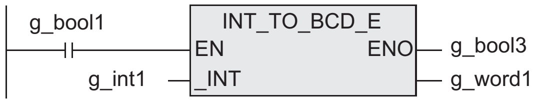


[ST]

```
g_word1 := INT_TO_BCD(g_int1);
```

2) Function with EN/ENO(INT_TO_BCD_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_BCD_E(g_bool1, g_int1, g_word1);
```

5.1.20 DINT_TO_BCD(_E)

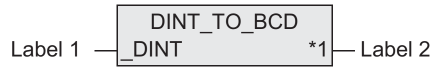
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

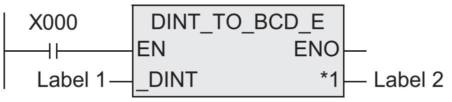
Outline

This function converts double word [signed] data into BCD data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_BCD		DINT_TO_BCD(_DINT);
		Example:
		Label 2:=
		DINT_TO_BCD(Label 1);

DINT_TO_BCD_E	 X000 DINT_TO_BCD_E EN ENO Label 1 _DINT *1 Label 2	DINT_TO_BCD_E(EN,_DINT, Output label); Example: DINT_TO_BCD_E(X000, Label 1, Label 2);

*1. Output variable

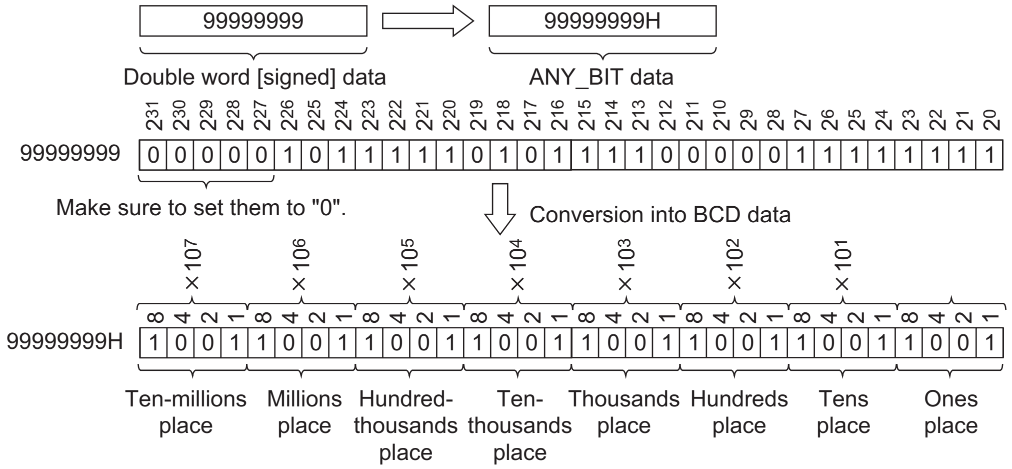
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	BCD data after conversion	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in (s) into BCD data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Error

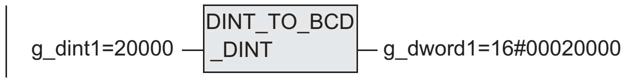
An operation error occurs when the value stored in a device specified in s is outside the range from "0" to "99,999,999".

Program example

In this program, double word [signed] data stored in a device specified in s is converted into BCD data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_BCD)

[Structured ladder]

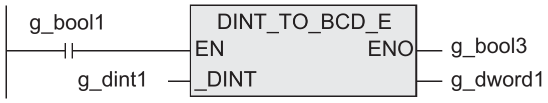


[ST]

```
g_dword1 := DINT_TO_BCD(g_dint1);
```

2) Function with EN/ENO(DINT_TO_BCD_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_BCD_E(g_bool1, g_dint1, g_dword1);
```

1 Outline
2 Function List
3 Function Construction
4 How to Read Explanation of Functions
5 Applied Functions
6 Standard Function Blocks
A Correspondence between Devices and Addresses

5.1.21 INT_TO_TIME(_E)

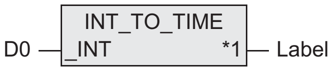
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

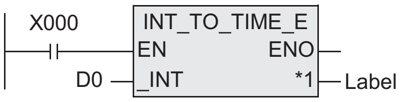
Outline

This function converts word [signed] data into time data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INT_TO_TIME		INT_TO_TIME(_INT);
		Example:
		Label:=
		INT_TO_TIME(D0);

INT_TO_TIME_E	 X000
	INT_TO_TIME_E
	EN
	ENO
	D0
	_INT
	*1
	Label
		INT_TO_TIME_E(EN,_INT,
		Output label);
		Example:
		INT_TO_TIME_E(X000,D0,Label);

*1. Output variable

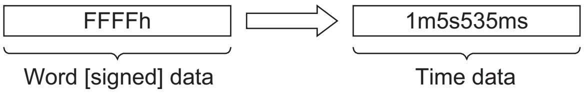
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_INT (s)	Conversion source word [signed] data	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [signed] data stored in a device specified in s into time data, and outputs the data obtained by conversion to a device specified in d.



Cautions

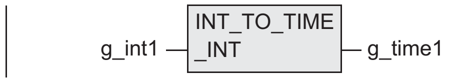
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, word [signed] data stored in a device specified in s is converted into time data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(INT_TO_TIME)

[Structured ladder]

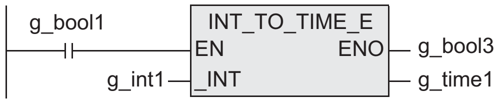


[ST]

```
g_time1 := INT_TO_TIME(g_int1);
```

2) Function with EN/ENO(INT_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := INT_TO_TIME_E(g_bool1, g_int1, g_time1);
```

5.1.22 DINT_TO_TIME(_E)

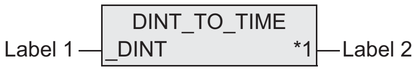
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

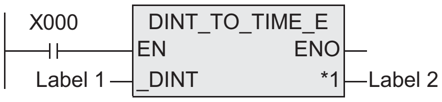
Outline

This function converts double word [signed] data into time data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DINT_TO_TIME		DINT_TO_TIME(_DINT);
		Example:
		Label 2:=
		DINT_TO_TIME(Label 1);

DINT_TO_TIME_
E
	 X000
	DINT_TO_TIME_E
	EN
	ENO
	Label 1
	_DINT
	*1
	Label 2
		DINT_TO_TIME_E(EN,_DINT, Output label);
		Example:
		DINT_TO_TIME_E(X000, Label 1, Label 2);

*1. Output variable

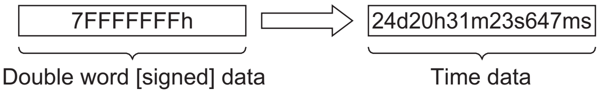
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DINT (s)	Conversion source double word [signed] data	Double Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [signed] data stored in a device specified in s into time data, and outputs the data obtained by conversion to a device specified in d.



Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

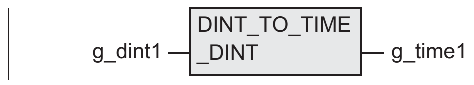
1 Outline
2 Function List
3 Function Construction
4 How to Read Explanation of Functions
5 Applied Functions
6 Standard Function Blocks
A Correspondence between Devices and Addresses

Program example

In this program, double word [signed] data stored in a device specified in s is converted into time data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DINT_TO_TIME)

[Structured ladder]

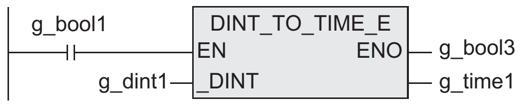


[ST]

```
g_time1 := DINT_TO_TIME(g_dint1);
```

2) Function with EN/ENO(DINT_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := DINT_TO_TIME_E(g_bool1, g_dint1, g_time1);
```

5.1.23 REAL_TO_INT(_E)

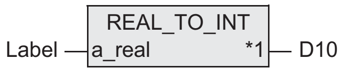
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	×	×	×	×	×

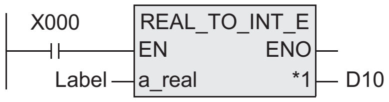
Outline

This function converts float (single precision) data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

REAL_TO_INT		REAL_TO_INT(a_real);
		Example:
		D10:=
		REAL_TO_INT(Label);

REAL_TO_INT_E	 X000 REAL_TO_INT_E EN ENO Label a_real *1 D10	REAL_TO_INT_E(EN,a_real, Output label); Example: REAL_TO_INT_E(X000, Label, D10);

*1. Output variable

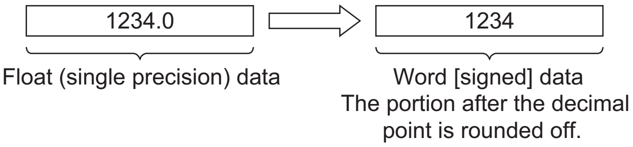
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_real (s)	Conversion source float (single precision) data	FLOAT (Single Precision)
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts float (single precision) data stored in a device specified in s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.


1234.0
1234
Float (single precision) data
Word [signed] data
The portion after the decimal point is rounded off.

Cautions

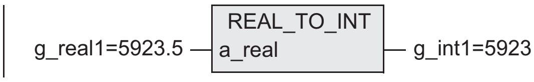
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.
4) In the data obtained by conversion, the portion after the decimal point of the float (single precision) data (source data) is rounded off.

Program example

In this program, float (single precision) data stored in a device specified in (s) is converted into word [signed] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(REAL_TO_INT)

[Structured ladder]

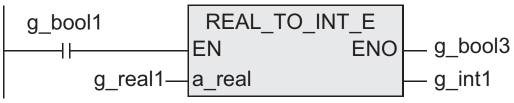


[ST]

g_int1 := REAL_TO_INT(g_real1);

2) Function with EN/ENO(REAL_TO_INT_E)

[Structured ladder]



[ST]

g_bool3 := REAL_TO_INT_E(g_bool1, g_real1, g_int1);

5.1.24 REAL_TO_DINT(_E)

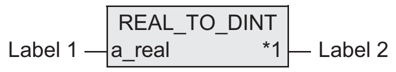
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	△	○	×	×	×	×	×

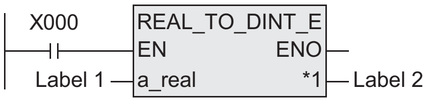
Outline

This function converts float (single precision) data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

REAL_TO_DINT		REAL_TO_DINT(a_real);
		Example:
		Label 2:=
		REAL_TO_DINT(Label 1);

REAL_TO_DINT_E	 X000
	REAL_TO_DINT_E
	EN
	ENO
	Label 1
	a_real
	*1
	Label 2
		REAL_TO_DINT_E(EN,a_real, Output label);
		Example:
		REAL_TO_DINT_E(X000, Label 1, Label 2);

*1. Output variable

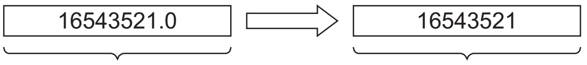
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	a_real (s)	Conversion source float (single precision) data	FLOAT (Single Precision)
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts float (single precision) data stored in a device specified in s into double word [signed] data, and outputs the data obtained by conversion to a device specified in d.



FLOAT (single precision) data

Double word [signed] data
The portion after the decimal
point is rounded off.

Cautions

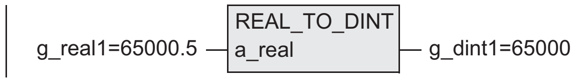
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
3) The function is provided in the FX3G Series Ver.1.10 or later.
4) In the data obtained by conversion, the portion after the decimal point of the float (single precision) data (source data) is rounded off.

Program example

In this program, float (single precision) data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(REAL_TO_DINT)

[Structured ladder]

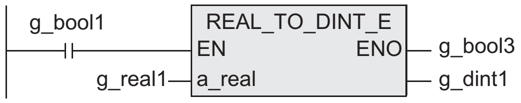


[ST]

```
g_dint1 := REAL_TO_DINT(g_real1);
```

2) Function with EN/ENO(DINT_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := REAL_TO_DINT_E(g_bool1, g_real1, g_dint1);
```

5.1.25 REAL_TO_STR(_E)

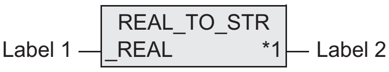
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

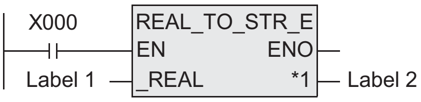
Outline

This function converts float (single precision) data into string data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

REAL_TO_STR		REAL_TO_STR(_REAL);
		Example:
		Label 2:=
		REAL_TO_STR(Label 1);

REAL_TO_STR_E	 X000 Label 1 REAL_TO_STR_E EN ENO _REAL *1 Label 2	REAL_TO_STR_E(EN,_REAL, Output label); Example: REAL_TO_STR_E(X000, Label 1, Label 2);

*1. Output variable

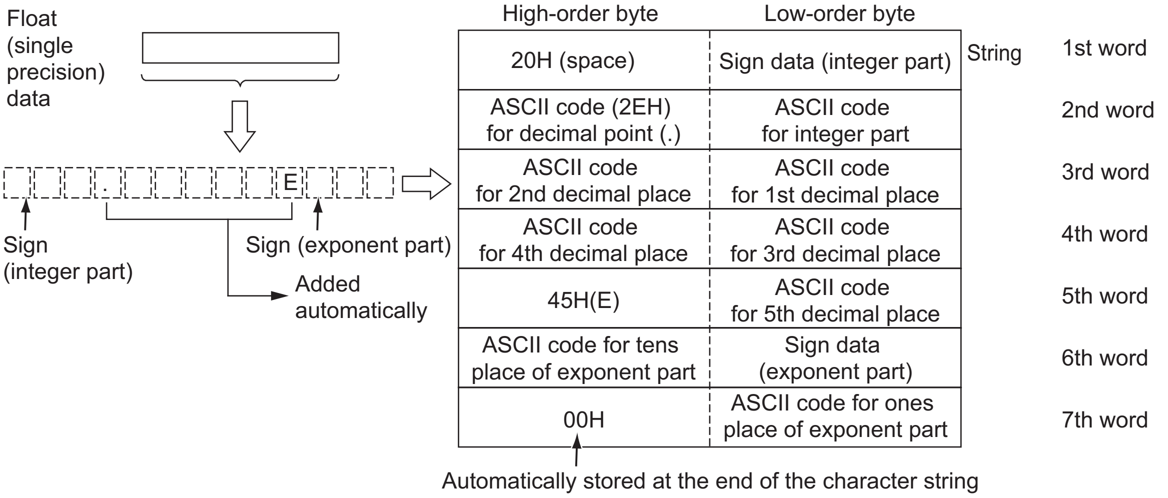
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_REAL (s)	Conversion source float (single precision) data	FLOAT (Single Precision)
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function converts float (single precision) data stored in a device specified in (s) into string (exponent) data, and outputs the data obtained by conversion to a device specified in (d).

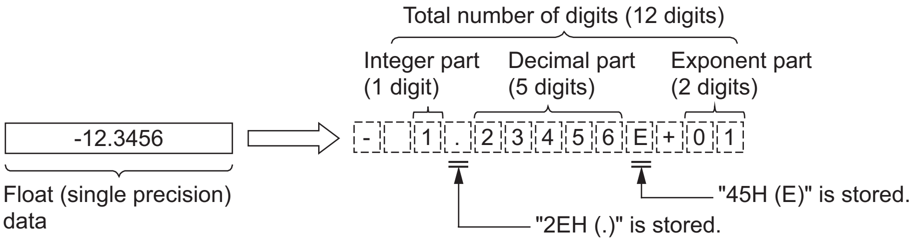


High-order byte	Low-order byte	
20H (space)	Sign data (integer part)	1st word
ASCII code (2EH) for decimal point (.)	ASCII code for integer part	2nd word
ASCII code for 2nd decimal place	ASCII code for 1st decimal place	3rd word
ASCII code for 4th decimal place	ASCII code for 3rd decimal place	4th word
45H(E)	ASCII code for 5th decimal place	5th word
ASCII code for tens place of exponent part	Sign data (exponent part)	6th word
00H	ASCII code for ones place of exponent part	7th word

Automatically stored at the end of the character string

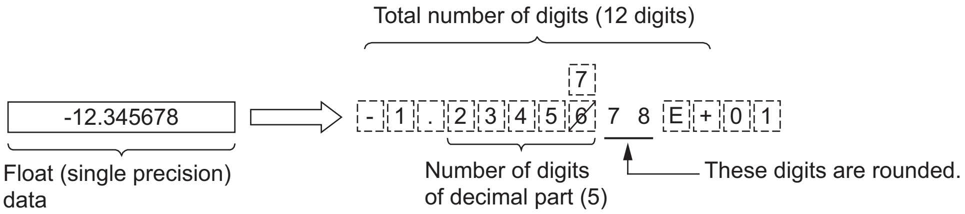
2) The string data obtained by conversion is output to a device specified in d as follows:

a) The number of digits is fixed respectively for the integer part, decimal part and exponent part as follows:
Integer part: 1, decimal part: 5, exponent part: 2
"2EH (.)" is automatically stored in the 3rd byte, and "45H (E)" is automatically stored in the 9th byte.

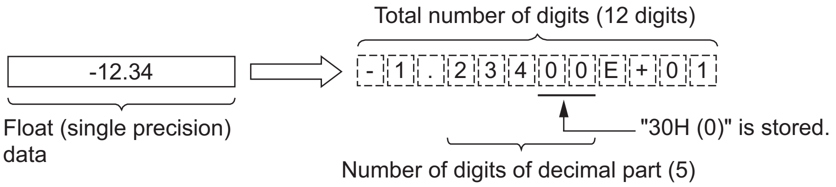


b) In "Sign data (integer part)", "20H (space)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

c) The 6th and later digits of the decimal part are rounded.

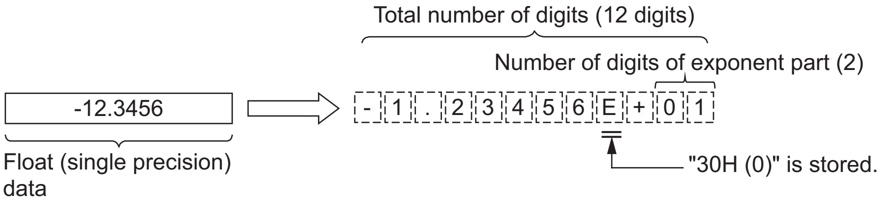

Total number of digits (12 digits)
7
-12.345678
- 1 . 2 3 4 5 6 7 8 E + 0 1
Float (single precision) data
Number of digits of decimal part (5)
These digits are rounded.

d) "30H (0)" is stored in the decimal part when the number of significant figures is small.


Total number of digits (12 digits)
-12.34
- 1 . 2 3 4 0 0 E + 0 1
Float (single precision) data
"30H (0)" is stored.
Number of digits of decimal part (5)

e) In "Sign data (exponent part)", "2BH (+)" is stored when the input value is positive, and "2DH (-)" is stored when the input value is negative.

f) "30H (0)" is stored in the tens place of the exponent part when the exponent part consists of 1 digit.


Total number of digits (12 digits)
Number of digits of exponent part (2)
-12.3456
- 1 . 2 3 4 5 6 E + 0 1
Float (single precision) data
"30H (0)" is stored.

3) "00H" is automatically stored at the end (7th word) of the character string.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

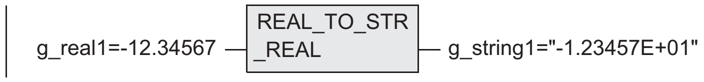
1) When the value stored in a device specified in s is outside the following range:
   0, $\pm 2^{-126} \leq$ (Value of device specified in s) $\leq \pm 2^{128}$
   (Error code: K6706)
2) When the range of a device which will store the character string obtained by conversion (device specified in d) exceeds the range of the corresponding device
   (Error code: K6706)
3) When the conversion result exceeds the specified total number of digits
   (Error code: K6706)

Program example

In this program, float (single precision) data stored in a device specified in s is converted into string data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(REAL_TO_STR)

[Structured ladder]

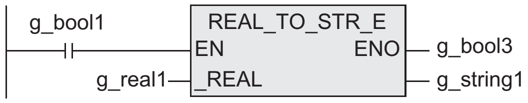


[ST]

```
g_string1 := REAL_TO_STR(g_real1);
```

2) Function with EN/ENO(REAL_TO_STR_E)

[Structured ladder]



[ST]

```
g_bool3 := REAL_TO_STR_E(g_bool1, g_real1, g_string1);
```

5.1.26 WORD_TO_BOOL(_E)

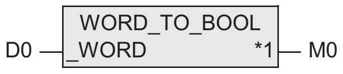
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

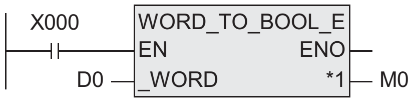
Outline

This function converts word [unsigned]/bit string [16-bit] data into bit data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

WORD_TO_BOOL		WORD_TO_BOOL(_WORD);
		Example:
		M0:=
		WORD_TO_BOOL(D0);

WORD_TO_BOOL_E	 X000
	D0
	WORD_TO_BOOL_E
	EN
	ENO
	_WORD
	*1
	M0
		WORD_TO_BOOL_E(EN, _WORD, Output label);
		Example:
		WORD_TO_BOOL_E(X000,D0, M0);

*1. Output variable

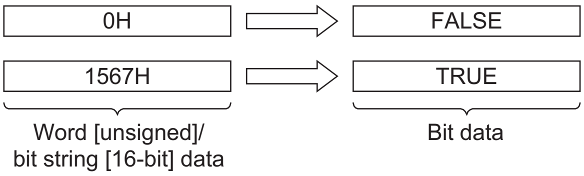
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit String [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in s into bit data, and outputs the data obtained by conversion to a device specified in d.



Cautions

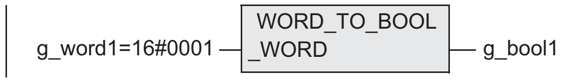
Use the function having "_E" in its name to connect a bus.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in s is converted into bit data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(WORD_TO_BOOL)

[Structured ladder]

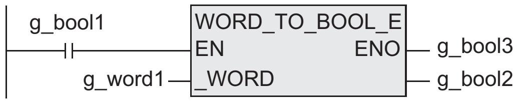


[ST]

```
g_bool1 := WORD_TO_BOOL(g_word1);
```

2) Function with EN/ENO(WORD_TO_BOOL_E)

[Structured ladder]



[ST]

```
g_bool3 := WORD_TO_BOOL_E(g_bool1, g_word1, g_bool2);
```

5.1.27 DWORD_TO_BOOL(_E)

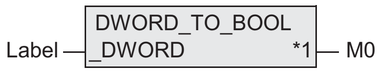
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

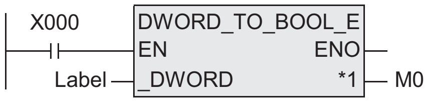
Outline

This function converts double word [unsigned]/bit string [32-bit] data into bit data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DWORD_TO_BOOL		DWORD_TO_BOOL(_DWORD);
		Example:
		M0:=
		DWORD_TO_BOOL(Label);

DWORD_TO_BOOL_E	 X000 Label DWORD_TO_BOOL_E EN ENO _DWORD *1 M0	DWORD_TO_BOOL_E(EN, _DWORD, Output label); Example: DWORD_TO_BOOL_E(X000, Label, M0);

*1. Output variable

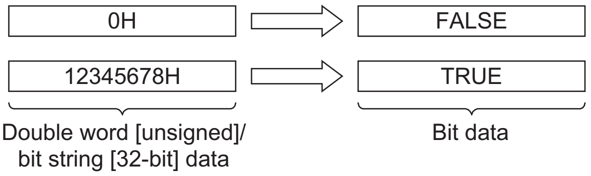
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

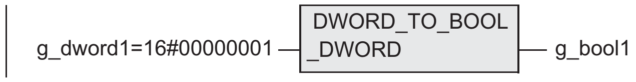
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in s is converted into bit data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DWORD_TO_BOOL)

[Structured ladder]

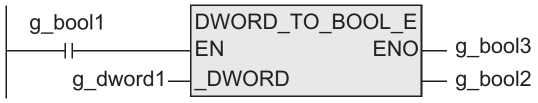


[ST]

```
g_bool1 := DWORD_TO_BOOL(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_BOOL_E)

[Structured ladder]



[ST]

```
g_bool3 := DWORD_TO_BOOL_E(g_bool1, g_dword1, g_bool2);
```

5.1.28 WORD_TO_INT(_E)

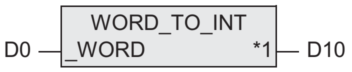
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

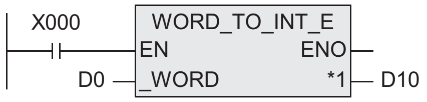
Outline

This function converts word [unsigned]/bit string [16-bit] data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

WORD_TO_INT		WORD_TO_INT(_WORD);
		Example:
		D10:=
		WORD_TO_INT(D0);

WORD_TO_INT_ E	 X000 WORD_TO_INT_E EN ENO D0 _WORD *1 D10	WORD_TO_INT_E(EN,_WORD, Output label); Example: WORD_TO_INT_E(X000,D0, D10);

*1. Output variable

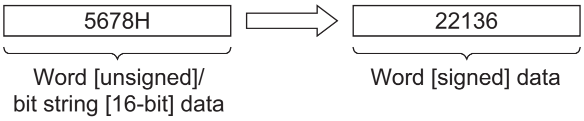
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

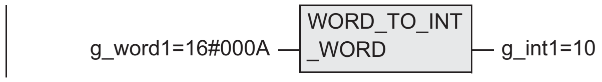
Use the function having "_E" in its name to connect a bus.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in s is converted into word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(WORD_TO_INT)

[Structured ladder]

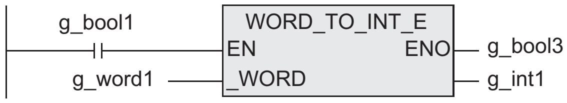


[ST]

```
g_int1 := WORD_TO_INT(g_word1);
```

2) Function with EN/ENO(WORD_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := WORD_TO_INT_E(g_bool1, g_word1, g_int1);
```

5.1.29 WORD_TO_DINT(_E)

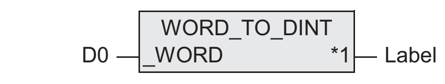
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

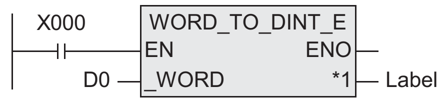
Outline

This function converts word [unsigned]/bit string [16-bit] data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

WORD_TO_DINT		WORD_TO_DINT(_WORD);
		Example:
		Label:=
		WORD_TO_DINT(D0);

WORD_TO_DINT _E	 X000 D0 WORD_TO_DINT_E EN ENO _WORD *1 Label	WORD_TO_DINT_E(EN,_WORD, Output label); Example: WORD_TO_DINT_E(X000,D0, Label);

*1. Output variable

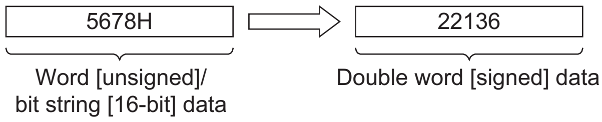
2. Set data

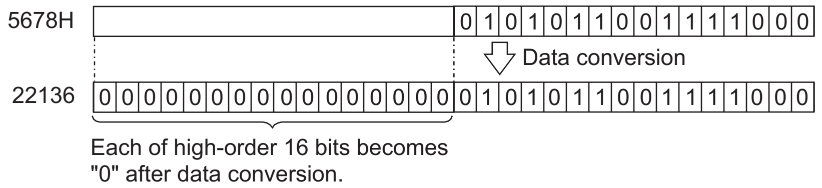
Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [unsigned]/bit string [16-bit] data storeds in a device specified in (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).





Cautions

1) Use the function having "_E" in its name to connect a bus.

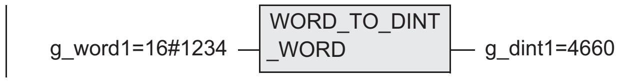
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(WORD_TO_DINT)

[Structured ladder]

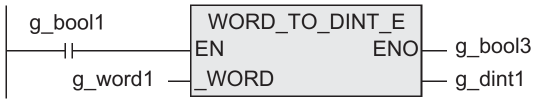


[ST]

```
g_dint1 := WORD_TO_DINT(g_word1);
```

2) Function with EN/ENO(WORD_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := WORD_TO_DINT_E(g_bool1, g_word1, g_dint1);
```

5.1.30 DWORD_TO_INT(_E)

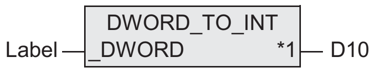
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

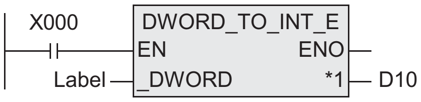
Outline

This function converts double word [unsigned]/bit string [32-bit] data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DWORD_TO_INT		DWORD_TO_INT(_DWORD);
		Example:
		D10:=
		DWORD_TO_INT(Label);

DWORD_TO_INT _E	 X000 Label DWORD_TO_INT_E EN ENO _DWORD *1 D10	DWORD_TO_INT_E(EN, _DWORD, Output label); Example: DWORD_TO_INT_E(X000,Label, D10);

*1. Output variable

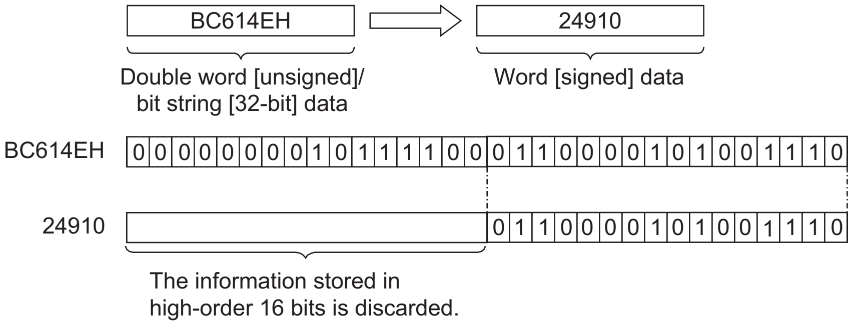
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in (s) into word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



BC614EH 0 0 0 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 1 0 0 1 1 1 0

24910 0 1 1 0 0 0 0 1 0 1 0 0 1 1 1 0

The information stored in
high-order 16 bits is discarded.

Cautions

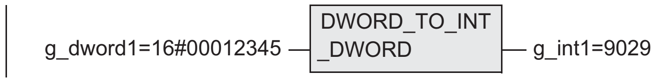
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in s is converted into word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DWORD_TO_INT)

[Structured ladder]

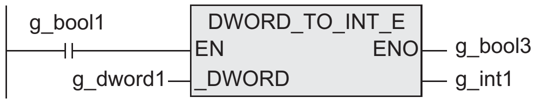


[ST]

```
g_int1 := DWORD_TO_INT(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := DWORD_TO_INT_E(g_bool1, g_dword1, g_int1);
```

5.1.31 DWORD_TO_DINT(_E)

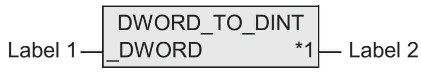
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

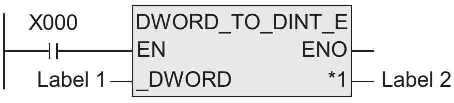
Outline

This function converts double word [unsigned]/bit string [32-bit] data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DWORD_TO_DINT		DWORD_TO_DINT(_DWORD);
		Example:
		Label 2:=
		DWORD_TO_DINT(Label 1);

DWORD_TO_DINT_E	 X000 DWORD_TO_DINT_E EN ENO Label 1 _DWORD *1 Label 2	DWORD_TO_DINT_E(EN, _DWORD, Output label); Example: DWORD_TO_DINT_E(X000, Label 1, Label 2);

*1. Output variable

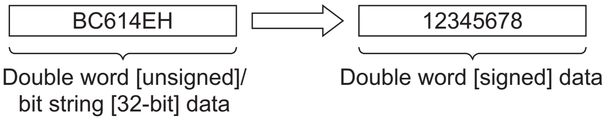
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (Ⓢ)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in Ⓢ into double word [signed] data, and outputs the data obtained by conversion to a device specified in ⓓ.


BC614EH
12345678
Double word [unsigned]/ bit string [32-bit] data
Double word [signed] data

Cautions

1) Use the function having "_E" in its name to connect a bus.

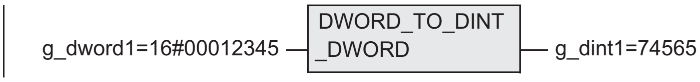
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DWORD_TO_DINT)

[Structured ladder]

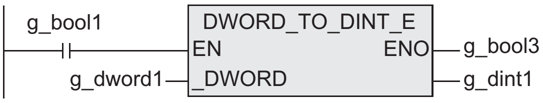


[ST]

```
g_dint1 := DWORD_TO_DINT(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := DWORD_TO_DINT_E(g_bool1, g_dword1, g_dint1);
```

5.1.32 WORD_TO_DWORD(_E)

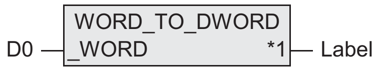
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

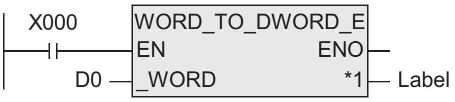
Outline

This function converts word [unsigned]/bit string [16-bit] data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

WORD_TO_DWORD		WORD_TO_DWORD(_WORD);
		Example:
		Label:=
		WORD_TO_DWORD(D0);

WORD_TO_DWORD_E	 X000 WORD_TO_DWORD_E EN ENO D0 _WORD *1 Label	WORD_TO_DWORD_E(EN, _WORD, Output label); Example: WORD_TO_DWORD_E(X000,D0, Label);

*1. Output variable

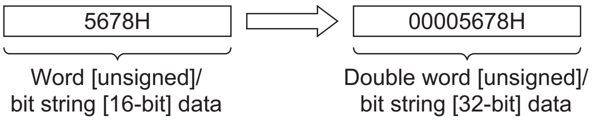
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word[unsigned]/bit string[32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in s into double word [unsigned]/bit [32-bit] data, and outputs the data obtained by conversion to a device specified in d. Each of high-order 16 bits becomes "0" after data conversion.



Cautions

1) Use the function having "_E" in its name to connect a bus.

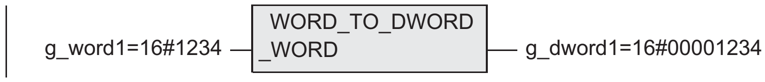
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in (s) is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(WORD_TO_DWORD)

[Structured ladder]

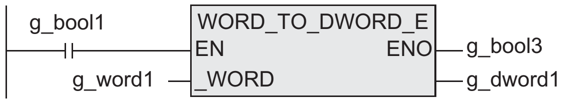


[ST]

```
g_dword1 := WORD_TO_DWORD(g_word1);
```

2) Function with EN/ENO(WORD_TO_DWORD_E)

[Structured ladder]



[ST]

```
g_bool3 := WORD_TO_DWORD_E(g_bool1, g_word1, g_dword1);
```

5.1.33 DWORD_TO_WORD(_E)

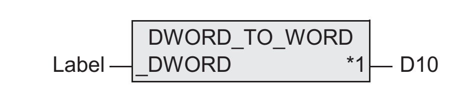
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

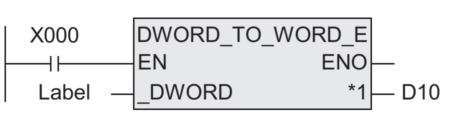
Outline

This function converts double word [unsigned]/bit string[32-bit] data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DWORD_TO_WORD		DWORD_TO_WORD(_DWORD);
		Example:
		D10:=
		DWORD_TO_WORD(Label);

DWORD_TO_WORD_E	 X000 Label DWORD_TO_WORD_E EN ENO _DWORD *1 D10	DWORD_TO_WORD_E(EN, _DWORD, Output label); Example: DWORD_TO_WORD_E(X000, Label, D10);

*1. Output variable

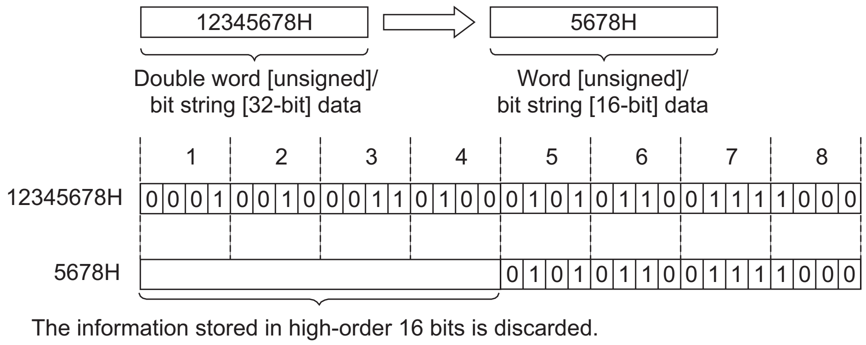
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned] /Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in (s) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

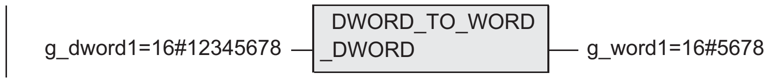
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in s is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(WORD_TO_DWORD)

[Structured ladder]

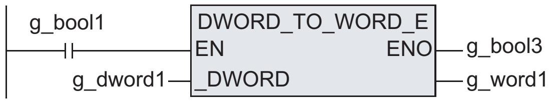


[ST]

```
g_word1 := DWORD_TO_WORD(g_dword1);
```

2) Function with EN/ENO(WORD_TO_DWORD_E)

[Structured ladder]



[ST]

```
g_bool3 := DWORD_TO_WORD_E(g_bool1, g_dword1, g_word1);
```

5.1.34 WORD_TO_TIME(_E)

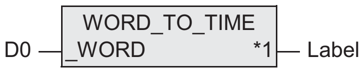
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

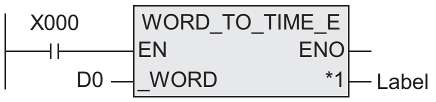
Outline

This function converts word [unsigned]/bit string [16-bit] data into time data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

WORD_TO_TIME		WORD_TO_TIME(_WORD);
		Example:
		Label:=
		WORD_TO_TIME(D0);

WORD_TO_TIME _E	 X000 WORD_TO_TIME_E EN ENO D0 _WORD *1 Label	WORD_TO_TIME_E(EN,_WORD, Output label); Example: WORD_TO_TIME_E(X000,D0, Label);

*1. Output variable

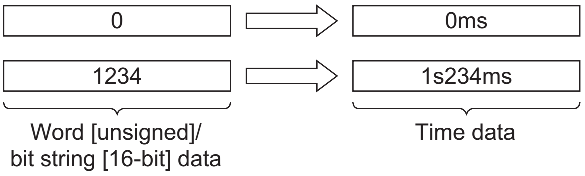
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_WORD (s)	Conversion source word [unsigned]/bit string [16-bit] data	Word [unsigned]/ Bit String[16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts word [unsigned]/bit string [16-bit] data stored in a device specified in s into time data, and outputs the data obtained by conversion to a device specified in d.



Cautions

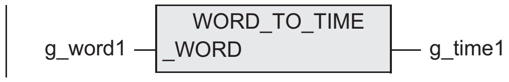
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in s is converted into time data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(WORD_TO_TIME)

[Structured ladder]

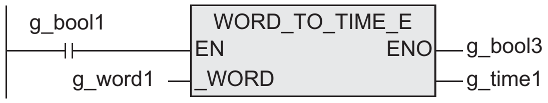


[ST]

```
g_time1 := WORD_TO_TIME(g_word1);
```

2) Function with EN/ENO(WORD_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := WORD_TO_TIME_E(g_bool1, g_word1, g_time1);
```

5.1.35 DWORD_TO_TIME(_E)

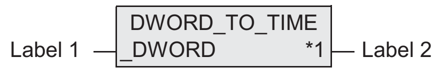
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

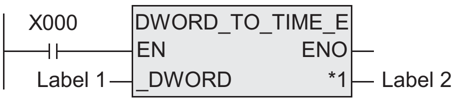
Outline

This function converts double word [unsigned]/bit string [32-bit] data into time data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DWORD_TO_TIME		DWORD_TO_TIME(_DWORD);
		Example:
		Label 2:=
		DWORD_TO_TIME(Label 1);

DWORD_TO_TIME_E	 X000
	DWORD_TO_TIME_E
	EN
	ENO
	Label 1
	_DWORD
	*1
	Label 2
		DWORD_TO_TIME_E(EN,_ DWORD, Output label);
		Example:
		DWORD_TO_TIME_E(X000, Label 1, Label 2);

*1. Output variable

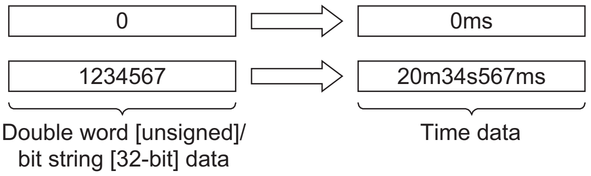
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_DWORD (s)	Conversion source double word [unsigned]/bit string [32-bit] data	Double Word [unsigned]/ Bit string [32-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts double word [unsigned]/bit string [32-bit] data stored in a device specified in s into time data, and outputs the data obtained by conversion to a device specified in d.



Cautions

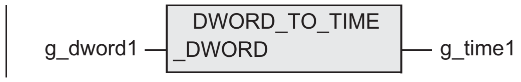
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, double word [unsigned]/bit string [32-bit] data stored in a device specified in s is converted into time data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(DWORD_TO_TIME)

[Structured ladder]

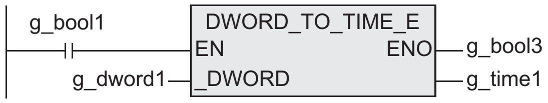


[ST]

```
g_time1 := DWORD_TO_TIME(g_dword1);
```

2) Function with EN/ENO(DWORD_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := DWORD_TO_TIME_E(g_bool1, g_dword1, g_time1);
```

5.1.36 STR_TO_BOOL(_E)

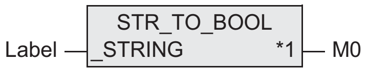
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

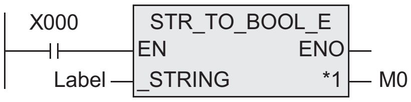
Outline

This function converts string data into bit data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

STR_TO_BOOL		STR_TO_BOOL(_STRING);
		Example:
		M0:=
		STR_TO_BOOL(Label);

STR_TO_BOOL_E	 X000
	STR_TO_BOOL_E
	EN
	ENO
	Label
	_STRING
	*1
	M0
		STR_TO_BOOL_E(EN,_STRING, Output label);
		Example:
		STR_TO_BOOL_E(X000, Label, M0);

*1. Output variable

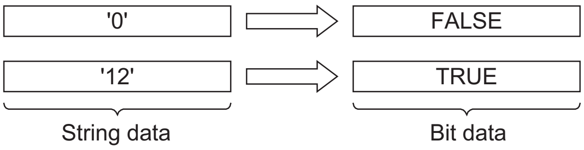
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts string data stored in a device specified in (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

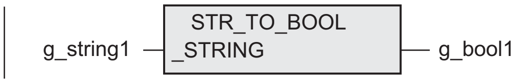
1) Use the function having "_E" in its name to connect a bus.
2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data.
Use global labels when specifying labels.

Program example

In this program, string data stored in a device specified in s is converted into bit data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(STR_TO_BOOL)

[Structured ladder]

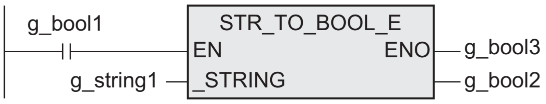


[ST]

```
g_bool1 := STR_TO_BOOL(g_string1);
```

2) Function with EN/ENO(STR_TO_BOOL_E)

[Structured ladder]



[ST]

```
g_bool3 := STR_TO_BOOL_E(g_bool1, g_string1, g_bool2);
```

5.1.37 STR_TO_INT(_E)

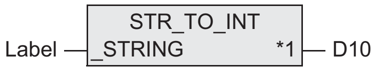
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

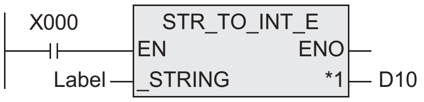
Outline

This function converts string data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

STR_TO_INT		STR_TO_INT(_STRING);
		Example:
		D10:=
		STR_TO_INT(Label);

STR_TO_INT_E	 X000 STR_TO_INT_E EN ENO Label _STRING *1 D10	STR_TO_INT_E(EN,_STRING, Output label); Example: STR_TO_INT_E(X000, Label, D10);

*1. Output variable

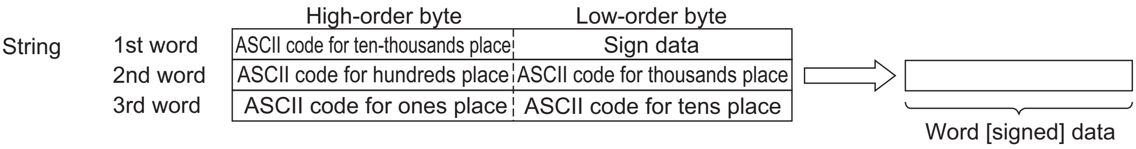
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts string data (3 words) stored in a device specified in s into word [signed] data, and outputs the data obtained by conversion to a device specified in d.



String		High-order byte	Low-order byte
	1st word	ASCII code for ten-thousands place	Sign data
	2nd word	ASCII code for hundreds place	ASCII code for thousands place
	3rd word	ASCII code for ones place	ASCII code for tens place

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data.
Use global labels when specifying labels.

Error

1) When the sign data (low-order byte) of the 1st word stored in a device specified in s is any other than "20H (space)" or "2DH (-)"
(Error code: K6706)

2) When the ASCII code for each place (digit) stored in s to s+2 is any other than "30H" to "39H", "20H (space)" or "00H (NULL)"
(Error code: K6706)

3) When the value stored in s to s+2 is outside the following range:
-32768 to +32767
(Error code: K6706)

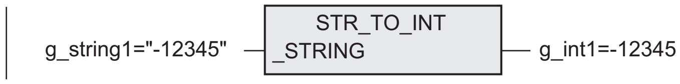
4) When any of devices s to s+2 exceeds the device range
(Error code: K6706)

Program example

In this program, string data stored in a device specified in s is converted into word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(STR_TO_INT)

[Structured ladder]

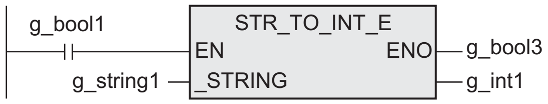


[ST]

```
g_int1 := STR_TO_INT(g_string1);
```

2) Function with EN/ENO(STR_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := STR_TO_INT_E(g_bool1, g_string1, g_int1);
```

5.1.38 STR_TO_DINT(_E)

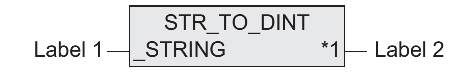
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

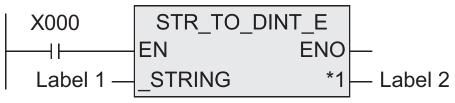
Outline

This function converts string data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

STR_TO_DINT		STR_TO_DINT(_STRING);
		Example:
		Label 2:=
		STR_TO_DINT(Label 1);

STR_TO_DINT_E	 X000 STR_TO_DINT_E EN ENO Label 1 _STRING *1 Label 2	STR_TO_DINT_E(EN,_STRING, Output label); Example: STR_TO_DINT_E(X000, Label 1, Label 2);

*1. Output variable

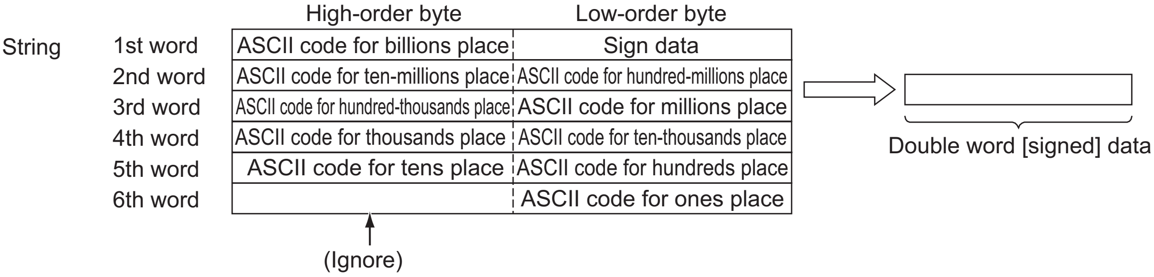
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts string data (6 words) stored in a device specified in (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



		High-order byte	Low-order byte
String	1st word	ASCII code for billions place	Sign data
	2nd word	ASCII code for ten-millions place	ASCII code for hundred-millions place
	3rd word	ASCII code for hundred-thousands place	ASCII code for millions place
	4th word	ASCII code for thousands place	ASCII code for ten-thousands place
	5th word	ASCII code for tens place	ASCII code for hundreds place
	6th word		ASCII code for ones place

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

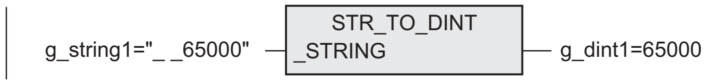
1) When the sign data (low-order byte) of the 1st word stored in a device specified in s is any other than "20H (space)" or "2DH (-)"
   (Error code: K6706)
2) When the ASCII code for each place (digit) stored in s to s+5 is any other than "30H" to "39H", "20H (space)" or "00H (NULL)"
   (Error code: K6706)
3) When the value stored in s to s+5 is outside the following range:
   -2,147,483,648 to +2,147,483,647
   (Error code: K6706)
4) When any of devices s to s+5 exceeds the device range
   (Error code: K6706)

Program example

In this program, string data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(STR_TO_DINT)

[Structured ladder]

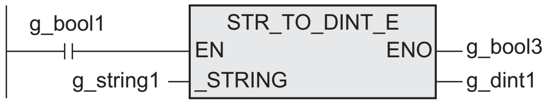


[ST]

```
g_dint1 := STR_TO_DINT(g_string1);
```

2) Function with EN/ENO(STR_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := STR_TO_DINT_E(g_bool1, g_string1, g_dint1);
```

5.1.39 STR_TO_REAL(_E)

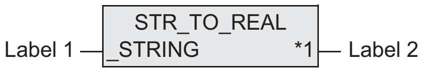
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

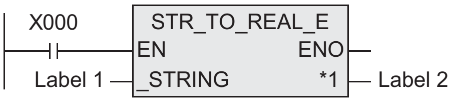
Outline

This function converts string data into float (single precision) data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

STR_TO_REAL		STR_TO_REAL(_STRING);
		Example:
		Label 2:=
		STR_TO_REAL(Label 1);

STR_TO_REAL_E	 X000 STR_TO_REAL_E EN ENO Label 1 _STRING *1 Label 2	STR_TO_REAL_E(EN,_STRING, Output label); Example: STR_TO_REAL_E(X000, Label 1, Label 2);

*1. Output variable

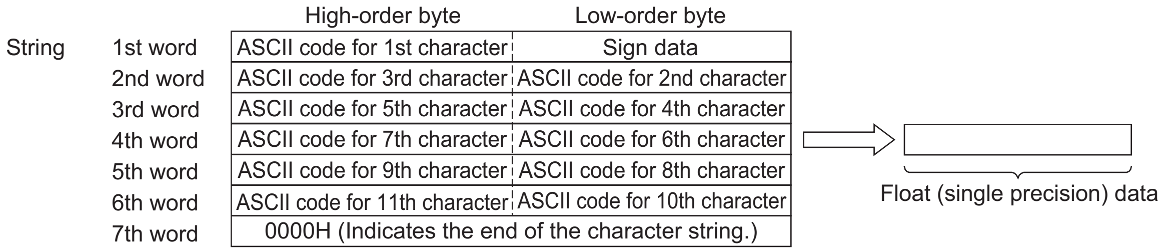
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Float (single precision) data after conversion	FLOAT (Single Precision)

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

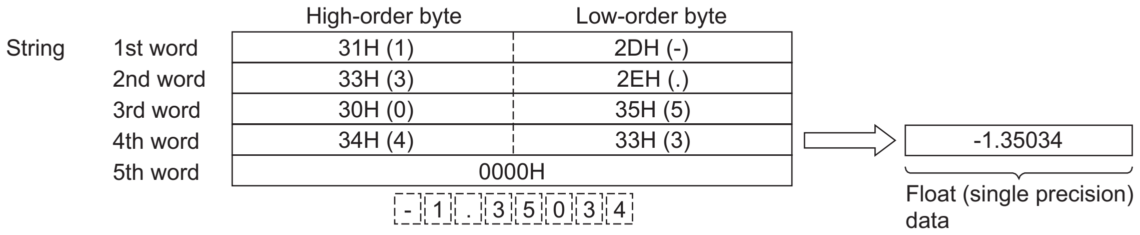
1) This function converts string data (in the decimal format or exponent format) stored in a device specified in (s) into float (single precision) data, and outputs the data obtained by conversion to a device specified in (d).



String	High-order byte	Low-order byte
1st word	ASCII code for 1st character	Sign data
2nd word	ASCII code for 3rd character	ASCII code for 2nd character
3rd word	ASCII code for 5th character	ASCII code for 4th character
4th word	ASCII code for 7th character	ASCII code for 6th character
5th word	ASCII code for 9th character	ASCII code for 8th character
6th word	ASCII code for 11th character	ASCII code for 10th character
7th word	0000H (Indicates the end of the character string.)	

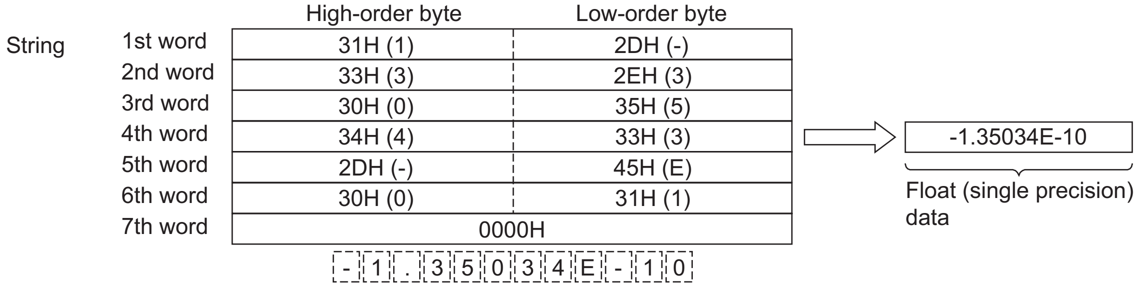
2) The conversion source string data can be in the decimal format or exponent format.

a) In the case of decimal format



String	High-order byte	Low-order byte
1st word	31H (1)	2DH (-)
2nd word	33H (3)	2EH (.)
3rd word	30H (0)	35H (5)
4th word	34H (4)	33H (3)
5th word	0000H	

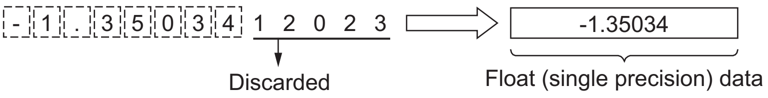
b) In the case of exponent format



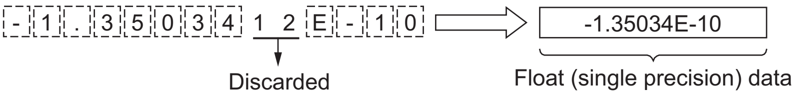
String	High-order byte	Low-order byte
1st word	31H (1)	2DH (-)
2nd word	33H (3)	2EH (3)
3rd word	30H (0)	35H (5)
4th word	34H (4)	33H (3)
5th word	2DH (-)	45H (E)
6th word	30H (0)	31H (1)
7th word	0000H	

3) With regard to string data, six digits excluding the sign, decimal point and exponent part are valid, and the 7th and later digits are discarded during conversion.

a) In the case of decimal format



b) In the case of exponent format

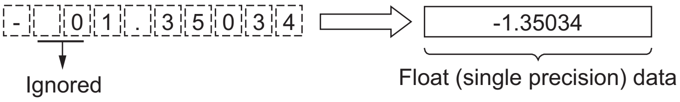


4) String data in the decimal format is handled as positive value during conversion when the sign is set to "2BH (+)" or when the sign is omitted. It is handled as negative value during conversion when the sign is set to "2DH (-)".

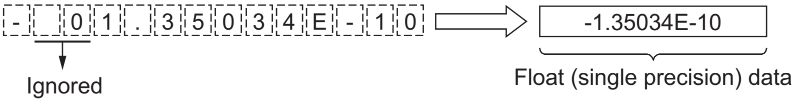
5) String data in the exponent format is handled as positive value during conversion when the sign of the exponent part is set to "2BH (+)" or when the sign is omitted. It is handled as negative value during conversion when the sign is set to "2DH (-)".

6) When "20H (space)" or "30H (0)" exists between the sign and the first number except "0" in string data, "20H (space)" or "30H (0)" is ignored during conversion.

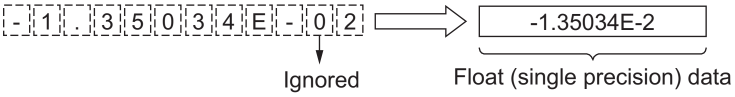
a) In the case of decimal format



b) In the case of exponent format



7) When "30H (0)" exists between "E" and a number in character string data (in the exponent format), "30H (0)" is ignored during conversion.



8) When "20H (space)" is contained in character string, "20H (space)" is ignored during conversion.

9) Up to 24 characters can be input as string data.
Each of "20H (space)" and "30H (0)" contained in string is counted as 1 character respectively.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When any character other than "30H (0)" to "39H (9)" exists in the integer or decimal part
(Error code: K6706)

2) When "2EH (.)" exists in two or more positions inside the character string specified in s
(Error code: K6706)

3) When any character other than "45H (E)", "2BH (+)" or "2DH (-)" exists in the exponent part, or when two or more exponent parts exist
(Error code: K6706)

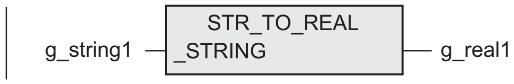
4) When the number of characters after s is "0" or any value larger than "24"
(Error code: K6706)

Program example

In this program, string data stored in a device specified in s is converted into float (single precision) data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(STR_TO_REAL)

[Structured ladder]

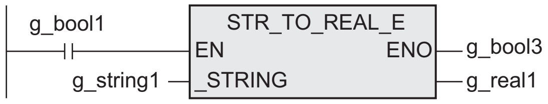


[ST]

```
g_real1 := STR_TO_REAL(g_string1);
```

2) Function with EN/ENO(STR_TO_REAL_E)

[Structured ladder]



[ST]

```
g_bool3 := STR_TO_REAL_E(g_bool1, g_string1, g_real1);
```

5.1.40 STR_TO_TIME(_E)

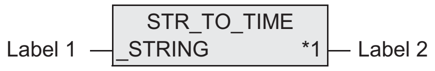
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

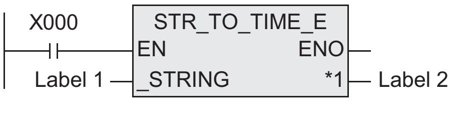
Outline

This function converts string data into time data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

STR_TO_TIME		STR_TO_TIME(_STRING);
		Example:
		Label 2:=
		STR_TO_TIME(Label 1);

STR_TO_TIME_E	 X000
	STR_TO_TIME_E
	EN
	ENO
	Label 1
	_STRING
	*1
	Label 2
		STR_TO_TIME_E(EN,_STRING, Output label);
		Example:
		STR_TO_TIME_E(X000, Label 1, Label 2);

*1. Output variable

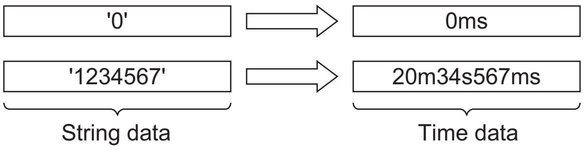
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_STRING (s)	Conversion source string data	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Time data after conversion	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts string data stored in a device specified in (s) into time data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling string data and 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling string data and 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the sign data of numeric data specified in s is any other than "20H (space)" or "2DH (-)"
(Error code: K6706)

2) When the ASCII code for each digit of character string data specified in s is any other than "30H (0)" to "39H (9)", "20H (space)" or "00H (NULL)"
(Error code: K6706)

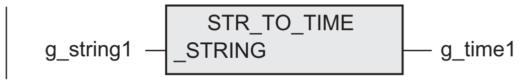
3) When the numeric value specified in s is outside the following range:
-2,147,483,648 to +2,147,483,647
(Error code: K6706)

Program example

In this program, string data stored in a device specified in s is converted into time data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(STR_TO_TIME)

[Structured ladder]

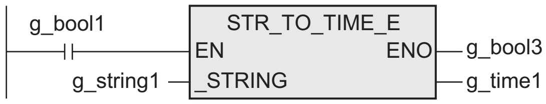


[ST]

```
g_time1 := STR_TO_TIME(g_string1);
```

2) Function with EN/ENO(STR_TO_TIME_E)

[Structured ladder]



[ST]

```
g_bool3 := STR_TO_TIME_E(g_bool1, g_string1, g_time1);
```

5.1.41 BCD_TO_INT(_E)

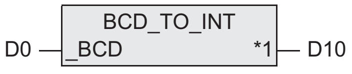
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

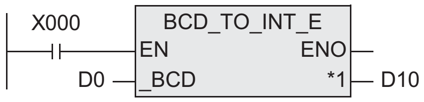
Outline

This function converts BCD data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BCD_TO_INT		BCD_TO_INT(_BCD);
		Example:
		D10:=
		BCD_TO_INT(D0);

BCD_TO_INT_E	 X000
	BCD_TO_INT_E
	EN
	ENO
	D0
	_BCD
	*1
	D10
		BCD_TO_INT_E(EN,_BCD,
		Output label);
		Example:
		BCD_TO_INT_E(X000,D0,D10);

*1. Output variable

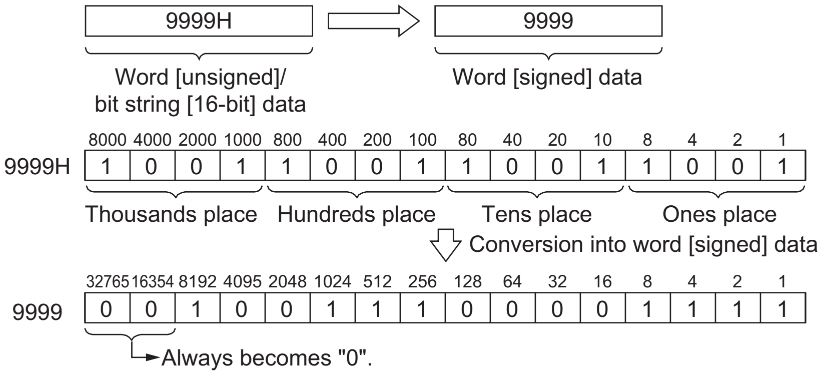
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BCD (s)	Conversion source BCD data	Word [unsigned]/ Bit String [16-bit]
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts BCD data stored in a device specified in (s) into word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



	8000	4000	2000	1000	800	400	200	100	80	40	20	10	8	4	2	1
9999H	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

	32765	16354	8192	4095	2048	1024	512	256	128	64	32	16	8	4	2	1
9999	0	0	1	0	0	1	1	1	0	0	0	0	1	1	1	1

Cautions

Use the function having "_E" in its name to connect a bus.

Error

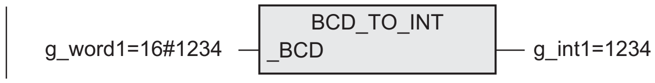
When the source data is not BCD (decimal number), M8067 (operation error) turns ON.

Program example

In this example, BCD data stored in a device specified in s is converted into word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(BCD_TO_INT)

[Structured ladder]

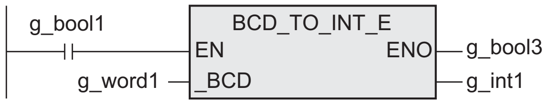


[ST]

```
g_int1 := BCD_TO_INT(g_word1);
```

2) Function with EN/ENO(BCD_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := BCD_TO_INT_E(g_bool1, g_word1, g_int1);
```

5.1.42 BCD_TO_DINT(_E)

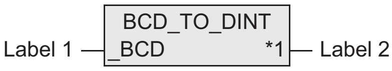
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

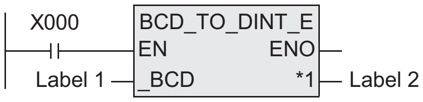
Outline

This function converts BCD data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

BCD_TO_DINT		BCD_TO_DINT(_BCD);
		Example:
		Label 2:=
		BCD_TO_DINT(Label 1);

BCD_TO_DINT_E	 X000 BCD_TO_DINT_E EN ENO Label 1 _BCD *1 Label 2	BCD_TO_DINT_E(EN,_BCD, Output label); Example: BCD_TO_DINT_E(X000, Label 1, Label 2);

*1. Output variable

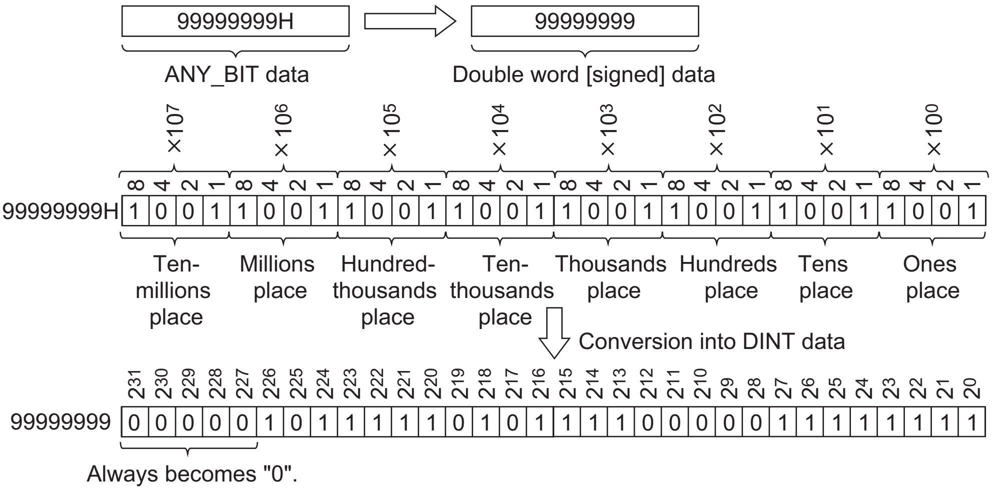
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_BCD (s)	Conversion source BCD data	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts BCD data stored in a device specified in (s) into double word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

Use the function having "_E" in its name to connect a bus.

Error

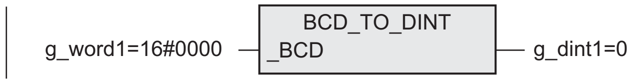
When the source data is not BCD (decimal number), M8067 (operation error) turns ON.

Program example

In this example, BCD data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(BCD_TO_DINT)

[Structured ladder]

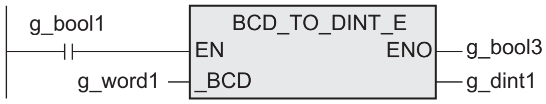


[ST]

```
g_dint1 := BCD_TO_DINT(g_word1);
```

2) Function with EN/ENO(BCD_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := BCD_TO_DINT_E(g_bool1, g_word1, g_dint1);
```

5.1.43 TIME_TO_BOOL(_E)

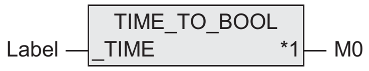
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

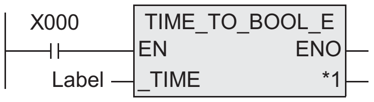
Outline

This function converts time data into bit data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TIME_TO_BOOL		TIME_TO_BOOL(_TIME);
		Example:
		M0:=
		TIME_TO_BOOL(Label);

TIME_TO_BOOL_E	 X000 TIME_TO_BOOL_E EN ENO Label _TIME *1 M0	TIME_TO_BOOL_E(EN,_TIME, Output label); Example: TIME_TO_BOOL_E(X000, Label, M0);

*1. Output variable

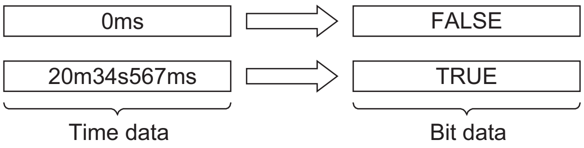
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Bit data after conversion	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in (s) into bit data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

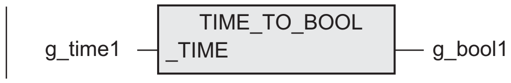
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, time data stored in a device specified in s is converted into bit data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(TIME_TO_BOOL)

[Structured ladder]

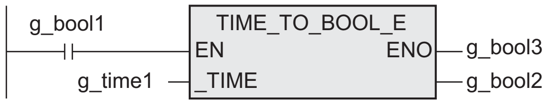


[ST]

g_bool1 := TIME_TO_BOOL(g_time1);

2) Function with EN/ENO(TIME_TO_BOOL_E)

[Structured ladder]



[ST]

g_bool3 := TIME_TO_BOOL_E(g_bool1, g_time1, g_bool2);

5.1.44 TIME_TO_INT(_E)

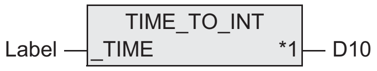
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

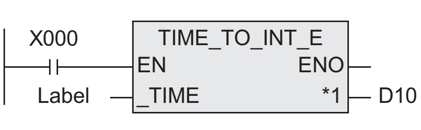
Outline

This function converts time data into word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TIME_TO_INT		TIME_TO_INT(_TIME);
		Example:
		D10:=
		TIME_TO_INT(Label);

TIME_TO_INT_E	 X000 TIME_TO_INT_E EN ENO Label _TIME *1 D10	TIME_TO_INT_E(EN,_TIME, Output label); Example: TIME_TO_INT_E(X000, Label, D10);

*1. Output variable

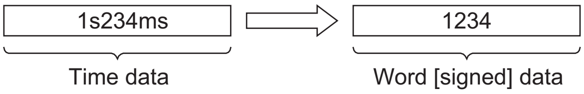
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [signed] data after conversion	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in (s) into word [signed] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

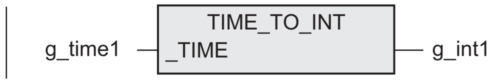
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, time data stored in a device specified in s is converted into word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(TIME_TO_INT)

[Structured ladder]

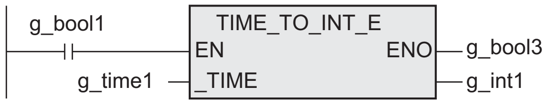


[ST]

```
g_int1 := TIME_TO_INT(g_time1);
```

2) Function with EN/ENO(TIME_TO_INT_E)

[Structured ladder]



[ST]

```
g_bool3 := TIME_TO_INT_E(g_bool1, g_time1, g_int1);
```

5.1.45 TIME_TO_DINT(_E)

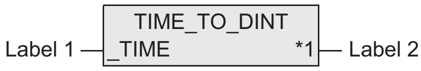
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

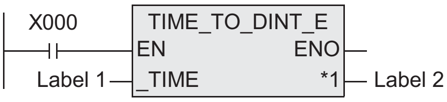
Outline

This function converts time data into double word [signed] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TIME_TO_DINT		TIME_TO_DINT(_TIME);
		Example:
		Label 2:=
		TIME_TO_DINT(Label 1);

TIME_TO_DINT_E	 X000
	TIME_TO_DINT_E
	EN
	ENO
	Label 1
	_TIME
	*1
	Label 2
		TIME_TO_DINT_E(EN,_TIME, Output label);
		Example:
		TIME_TO_DINT_E(X000,Label 1, Label 2);

*1. Output variable

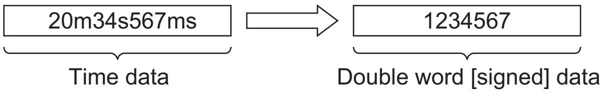
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (ⓢ)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Double word [signed] data after conversion	Double Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in ⓢ into double word [signed] data, and outputs the data obtained by conversion to a device specified in ⓓ.



Cautions

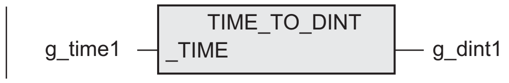
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, time data stored in a device specified in s is converted into double word [signed] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(TIME_TO_DINT)

[Structured ladder]

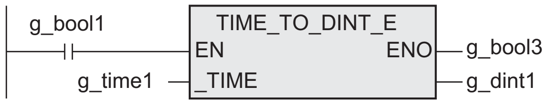


[ST]

```
g_dint1 := TIME_TO_DINT(g_time1);
```

2) Function with EN/ENO(TIME_TO_DINT_E)

[Structured ladder]



[ST]

```
g_bool3 := TIME_TO_DINT_E(g_bool1, g_time1, g_dint1);
```

5.1.46 TIME_TO_STR(_E)

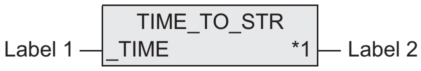
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

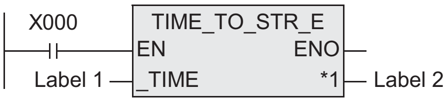
Outline

This function converts time data into string data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TIME_TO_STR		TIME_TO_STR(_TIME);
		Example:
		Label 2:=
		TIME_TO_STR(Label 1);

TIME_TO_STR_E	 X000 TIME_TO_STR_E EN ENO Label 1 _TIME *1 Label 2	TIME_TO_STR_E(EN,_TIME, Output label); Example: TIME_TO_STR_E(X000, Label 1, Label 2);

*1. Output variable

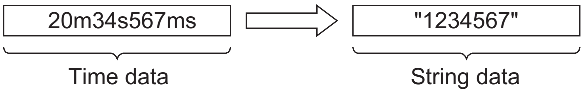
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	String data after conversion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in (s) into string data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Error

An operation error occurs in the following case. The error flag M8067 turns ON, and D8067 stores the error code.

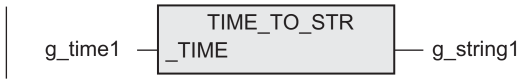
1) When the number of points occupied by the device specified in (d) exceeds the range of the corresponding device.

Program example

In this program, time data stored in a device specified in (s) is converted into string data, and the data obtained by conversion is output to a device specified in (d).

1) Function without EN/ENO(TIME_TO_STR)

[Structured ladder]

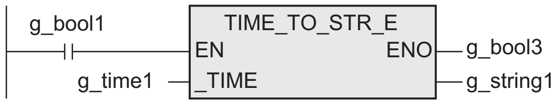


[ST]

```
g_string1 := TIME_TO_STR(g_time1);
```

2) Function with EN/ENO(TIME_TO_STR_E)

[Structured ladder]



[ST]

```
g_bool3 := TIME_TO_STR_E(g_bool1, g_time1, g_string1);
```

5.1.47 TIME_TO_WORD(_E)

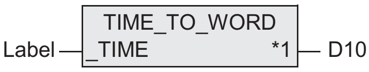
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

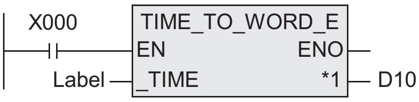
Outline

This function converts time data into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TIME_TO_WORD		TIME_TO_WORD(_TIME);
		Example:
		D10:=
		TIME_TO_WORD(Label);

TIME_TO_WORD _E	 X000 TIME_TO_WORD_E EN ENO Label _TIME *1 D10	TIME_TO_WORD_E(EN,_TIME, Output label); Example: TIME_TO_WORD_E(X000, Label, D10);

*1. Output variable

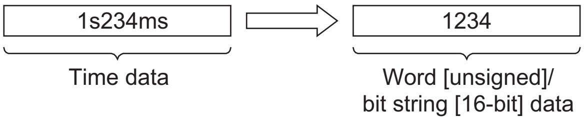
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Word [unsigned]/bit string [16-bit] data after conversion	Word [unsigned]/ Bit String [16-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in (s) into word [unsigned]/bit string [16-bit] data, and outputs the data obtained by conversion to a device specified in (d).



Cautions

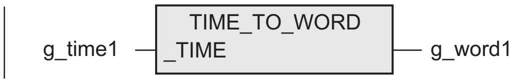
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, time data stored in a device specified in s is converted into word [unsigned]/bit string [16-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(TIME_TO_WORD)

[Structured ladder]

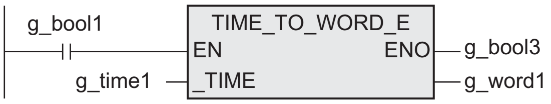


[ST]

```
g_word1 := TIME_TO_WORD(g_time1);
```

2) Function with EN/ENO(TIME_TO_WORD_E)

[Structured ladder]



[ST]

```
g_bool3 := TIME_TO_WORD_E(g_bool1, g_time1, g_word1);
```

5.1.48 TIME_TO_DWORD(_E)

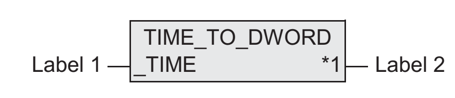
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

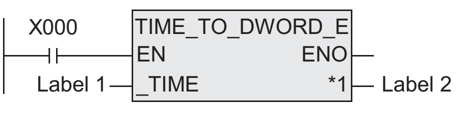
Outline

This function converts time data into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TIME_TO_DWORD		TIME_TO_DWORD(_TIME);
		Example:
		Label 2:=
		TIME_TO_DWORD(Label 1);

TIME_TO_DWORD_E	 X000
	Label 1
	TIME_TO_DWORD_E
	EN
	ENO
	_TIME
	*1
	Label 2
		TIME_TO_DWORD_E(EN,_TIME, Output label);
		Example:
		TIME_TO_DWORD_E(X000, Label 1, Label 2);

*1. Output variable

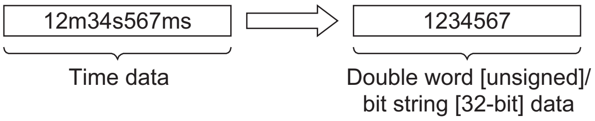
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_TIME (s)	Conversion source time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Double word [unsigned]/bit string [32-bit] data after conversion	Double Word [unsigned]/ Bit string [32-bit]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function converts time data stored in a device specified in s into double word [unsigned]/bit string [32-bit] data, and outputs the data obtained by conversion to a device specified in d.



Cautions

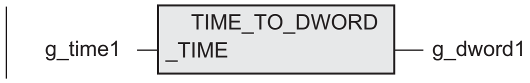
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, time data stored in a device specified in s is converted into double word [unsigned]/bit string [32-bit] data, and the data obtained by conversion is output to a device specified in d.

1) Function without EN/ENO(TIME_TO_DWORD)

[Structured ladder]

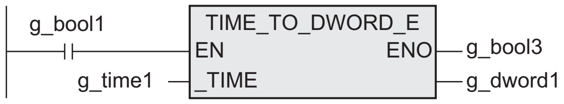


[ST]

```
g_dword1 := TIME_TO_DWORD(g_time1);
```

2) Function with EN/ENO(TIME_TO_DWORD_E)

[Structured ladder]



[ST]

```
g_dword1 := TIME_TO_DWORD(g_time1);
```

5.2 Standard Functions Of One Numeric Variable

5.2.1 ABS(_E)

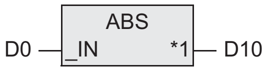
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

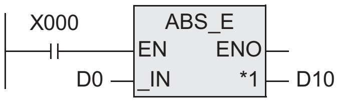
Outline

This function obtains the absolute value, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

ABS		ABS(_IN);
		Example:
		D10:=
		ABS(D0);

ABS_E		ABS_E(EN,_IN,Output label);
		Example:
		ABS_E(X000,D0,D10);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Data whose absolute value is to be obtained, or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the absolute value of word [signed]/double word [signed]/float (single precision) data stored in a device specified in s, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in a device specified in s.
This function is expressed as follows when the input value is "A" and the output operation result is "B".
B=|A|

2) When the data type stored in a device specified in s is word [signed] and the stored data is "-32768", this function outputs "-32768" to a device specified in d. (The maximum absolute value handled by this function is "32,767".)
When the data type stored in a device specified in s is double word [signed] and the stored data is "-2147483648", this function outputs "-2147483648" to a device specified in d. (The maximum absolute value handled by this function is "2147483647".)

Cautions

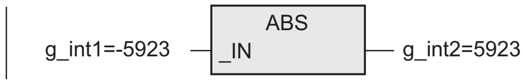
Use the function having "_E" in its name to connect a bus.

Program example

In this program, the absolute value is obtained for word [signed] data stored in a device specified in s, and the operation result is output to a device specified in d using the data type same as the data stored in a device specified in s.

1) Function without EN/ENO(ABS)

[Structured ladder]

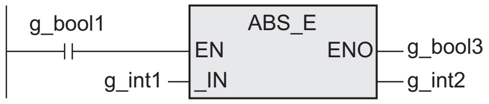


[ST]

```
g_int2 := ABS(g_int1);
```

2) Function with EN/ENO(ABS_E)

[Structured ladder]



[ST]

```
g_bool3 := ABS_E(g_bool1, g_int1, g_int2);
```

5.3 Standard Arithmetic Functions

5.3.1 ADD_E

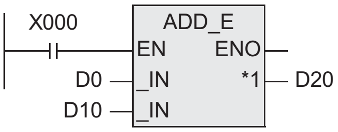
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function performs addition using two values (A + B = C), and outputs the operation result.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

ADD_E	 X000
	ADD_E
	EN
	ENO
	D0
	_IN
	*1
	D20
	D10
	_IN
		ADD_E(EN,_IN,_IN,Output label);
		Example:
		ADD_E(X000,D0,D10,D20);

*1. Output variable

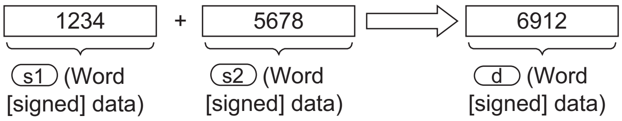
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s …)	Data for addition or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function performs addition (s1+s2) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]



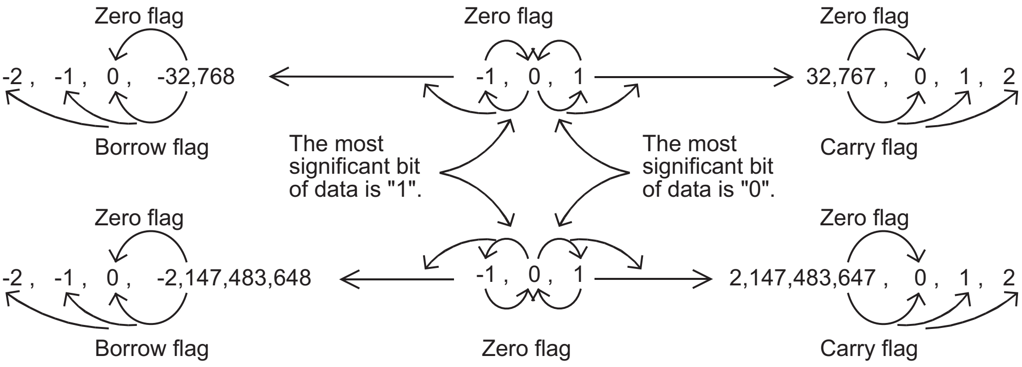
Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO.
However, note that the obtained operation result is not accurate in this case.

Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

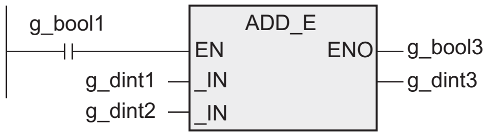
Device	Name	Description
M8020	Zero	ON : When the operation result is "0" OFF: When the operation result is any other than "0"
M8021	Borrow	ON : When the operation result is less than "-32,768" (16-bit operation) or less than "-2,147,483,648" (32-bit operation) OFF: When the operation result is "-32,768" (16-bit operation) or more or "-2,147,483,648" (32-bit operation) or more
M8022	Carry	ON : When the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation) OFF: When the operation result is "32,767" (16-bit operation) or less or "2,147,483,647" (32-bit operation) or less



Program example

In this program, addition is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=ADD_E(g_bool1,g_dint1,g_dint2,g_dint3);
```

5.3.2 SUB_E

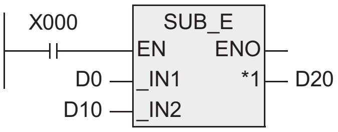
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function performs subtraction using two values (A - B = C), and outputs the operation result.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

SUB_E	 X000
	SUB_E
	EN
	ENO
	D0
	_IN1
	*1
	D20
	D10
	_IN2
		SUB_E(EN,_IN1,_IN2,Output label);
		Example:
		SUB_E(X000,D0,D10,D20);

*1. Output variable

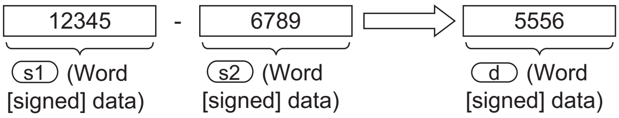
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Data to be subtracted or word device which stores such data	ANY_NUM
	_IN2 (s2)	Data for subtraction or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function performs subtraction (s1-s2) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]

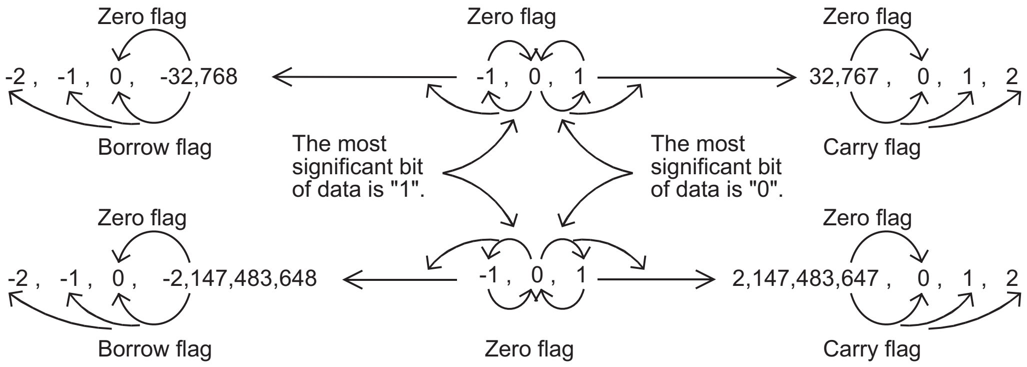


Cautions

1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO.
   However, note that the obtained operation result is not accurate in this case.

   Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

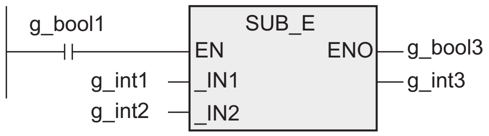
Device	Name	Description
M8020	Zero	ON : When the operation result is "0" OFF: When the operation result is any other than "0"
M8021	Borrow	ON : When the operation result is less than "-32,768" (16-bit operation) or less than "-2,147,483,648" (32-bit operation) OFF: When the operation result is "-32,768" (16-bit operation) or more or "-2,147,483,648" (32-bit operation) or more
M8022	Carry	ON : When the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation) OFF: When the operation result is "32,767" (16-bit operation) or less or "2,147,483,647" (32-bit operation) or less



Program example

In this program, subtraction is performed using word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=SUB_E(g_bool1,g_int1,g_int2,g_int3);
```

5.3.3 MUL_E

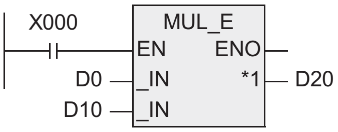
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function performs multiplication using two values (A × B = C), and outputs the operation result.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MUL_E	 X000
	MUL_E
	EN
	ENO
	D0
	_IN
	*1
	D20
	D10
	_IN
		MUL_E(EN,_IN,_IN,Output label);
		Example:
		MUL_E(X000,D0,D10,D20);

*1. Output variable

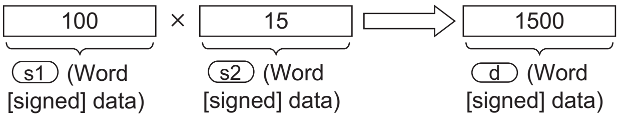
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 …)	Data for multiplication or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function performs multiplication (s1×s2) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]



Cautions

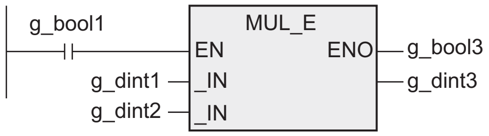
1) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

2) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error. "TRUE" is output from ENO.
However, note that the obtained operation result is not accurate in this case.

Program example

In this program, multiplication is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=MUL_E(g_bool1,g_dint1,g_dint2,g_dint3);
```

5.3.4 DIV_E

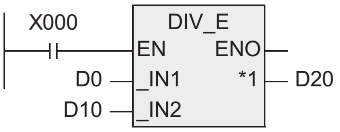
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function performs division using two values (A / B = C … remainder), and outputs the quotient.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DIV_E	 X000
	DIV_E
	EN
	ENO
	D0
	_IN1
	*1
	D20
	D10
	_IN2
		DIV_E(EN,_IN1,_IN2,Output label);
		Example:
		DIV_E(X000,D0,D10,D20);

*1. Output variable

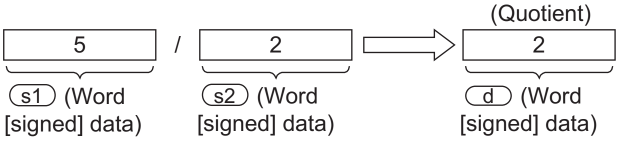
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Data to be divided, or word device which stores such data	ANY_NUM
	_IN2 (s2)	Data for division (divisor), or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_NUM

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function performs division (s1/s2) using word [signed]/double word [signed]/float (single precision) data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]



Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

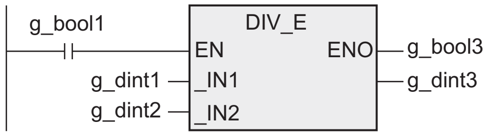
Error

1) An operation error occurs when the divisor stored in a device specified in s2 is "0", and the function is not executed.
2) An operation error occurs when the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation).

Program example

In this program, division is performed using double word [signed] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

[Structured ladder]



[ST]

```
g_bool3:=DIV_E(g_bool1,g_dint1,g_dint2,g_dint3);
```

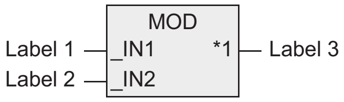
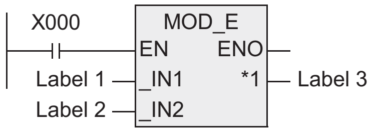
5.3.5 MOD(_E)

FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function performs division using two values (A / B = C … remainder), and outputs the remainder.

1. Format

Function name	Expression in each language	
	Structured ladder	ST
MOD		_IN1 MOD _IN2; *2 Example: Label 3:= Label 1 MOD Label 2;
MOD_E	 X000 MOD_E EN ENO Label 1 _IN1 *1 Label 3 Label 2 _IN2	MOD_E(EN,_IN1,_IN2,Output label; *2 Example: MOD_E(X000,Label 1, Label 2,Label 3);

*1. Output variable

*2. Refer to the Cautions

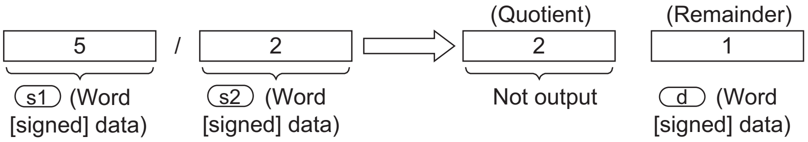
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Data to be divided, or word device which stores such data	ANY_INT
	_IN2 (s2)	Data for division (divisor), or word device which stores such data	ANY_INT
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	ANY_INT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function performs division (s1/s2) using word [signed]/double word [signed] data stored in devices specified in s1 and s2, and outputs the remainder to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]



Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
3) Note that the "MOD" description method is different from other function description methods in the ST language.

Error

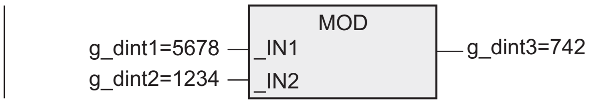
1) An operation error occurs when the divisor stored in a device specified in (s2) is "0", and the function is not executed.
2) An operation error occurs when the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation).

Program example

In this program, division is performed using double word [signed] data stored in devices specified in (s1) and (s2), and the remainder is output to a device specified in (d) using the data type of data stored in devices specified in (s1) and (s2).

1) Function without EN/ENO(MOD)

[Structured ladder]

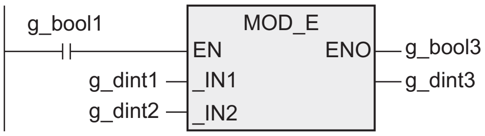


[ST]

```
g_dint3:=g_dint1 MOD g_dint2;
```

2) Function with EN/ENO(MOD_E)

[Structured ladder]]



[ST]

```
g_bool3 := MOD_E(g_bool1, g_dint1, g_dint2, g_dint3);
```

5.3.6 EXPT(_E)

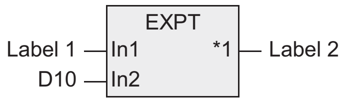
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

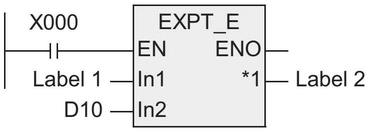
Outline

This function obtains raised result, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

EXPT		EXPT(In1,In2);
		Example:
		Label 2:=
		EXPT(Label 1,D10);

EXPT_E	 X000
	EXPT_E
	EN
	ENO
	Label 1
	In1
	*1
	Label 2
	D10
	In2
		EXPT_E(EN,In1,In2,Output label);
		Example:
		EXPT_E(X000,Label 1,D10, Label 2);

*1. Output variable

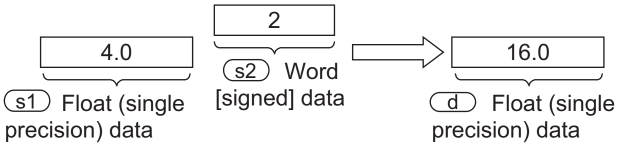
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	In1 (s1)	Data to be raised, or word device which stores such data	FLOAT (Single Precision)
	In2 (s2)	Power data, or word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the operation result	FLOAT (Single Precision)

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function raises float (single precision) data stored in a device specified in (s1) (to the power of the value stored in a device specified in (s2)), and outputs the operation result to a device specified in (d).


2
4.0
16.0
(s2) Word [signed] data
(s1) Float (single precision) data
(d) Float (single precision) data

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the value stored in a device specified in s1 is negative
(Error code: K6706)

2) When the value stored in a device specified in s1 is "0"
(Error code: K6706)

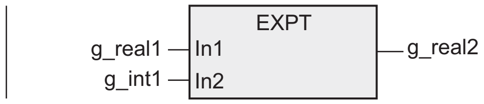
3) When the operation result is outside the following range:
(Error code: K6706)
$2^{-126} \leq |\text{Operation result}| < 2^{128}$

Program example

In this program, the value stored in a device specified in s1 is raised to the power of the value stored in a device specified in s2, and the operation result is output to a device specified in d using the data type of data stored in a device specified in s1.

1) Function without EN/ENO(EXPT)

[Structured ladder]

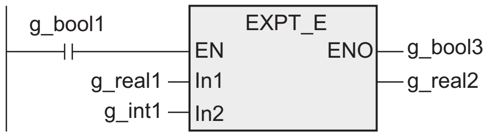


[ST]

```
g_real2:=EXPT(g_real1,g_int1);
```

2) Function with EN/ENO(EXPT_E)

[Structured ladder]



[ST]

```
g_bool3:=EXPT_E(g_bool1,g_real1,g_int1,g_real2);
```

5.3.7 MOVE(_E)

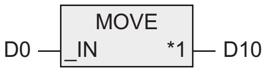
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

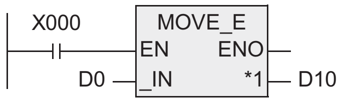
Outline

This function transfers data stored in a device to another device.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MOVE		MOVE(_IN);
		Example:
		D10:=
		MOVE(D0);

MOVE_E		MOVE_E(EN,_IN,Output label);
		Example:
		MOVE_E(X000,D0,D10);

*1. Output variable

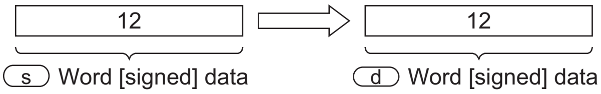
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Transfer source data, or word device which stores such data	ANY
Output variable	ENO	Execution status	Bit
	*1 (d)	Transfer destination word device	ANY

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function transfers data stored in a device specified in (s) to a device specified in (d).



Cautions

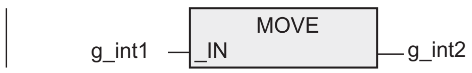
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, word [signed] data stored in a device specified in (s) is transferred to a device specified in (d).

1) Function without EN/ENO(MOVE)

[Structured ladder]

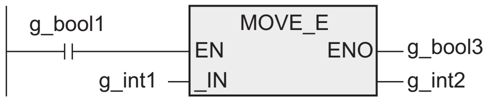


[ST]

g_int2:=MOVE(g_int1);

2) Function with EN/ENO(MOVE_E)

[Structured ladder]



[ST]

g_bool3:=MOVE_E(g_bool1,g_int1,g_int2);

5.4 Standard Bit Shift Functions

5.4.1 SHL(_E)

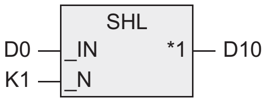
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

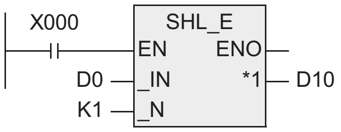
Outline

This function shifts data of specified bit length leftward by the specified number of bits.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

SHL		SHL(_IN,_N);
		Example:
		D10:=
		SHL(D0,K1);

SHL_E	 X000
	SHL_E
	EN
	ENO
	D0
	_IN
	*1
	D10
	K1
	_N
		SHL_E(EN,_IN,_N,Output label);
		Example:
		SHL_E(X000,D0,K1,D10);

*1. Output variable

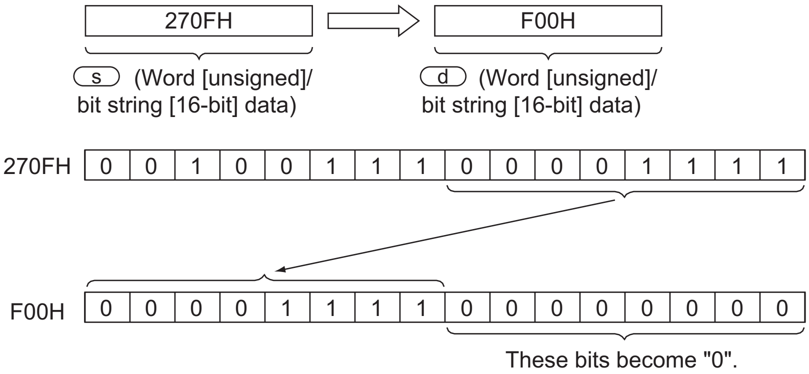
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Word device which stores data to be shifted leftward	ANY_BIT
	_N (n)	Number of shifted bits	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store data obtained by shift	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function shifts word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in s leftward by "n" bits, and outputs the obtained data to a device specified in d using the data type of data stored in a device specified in s.
Data is shifted leftward by "n" bits specified in n.
Example: When word [unsigned]/bit string [16-bit] data is stored in a device specified in s, and "8" is specified in n


270FH
F00H
s (Word [unsigned]/
bit string [16-bit] data)
d (Word [unsigned]/
bit string [16-bit] data)
270FH 0 0 1 0 0 1 1 1 0 0 0 0 1 1 1 1
F00H 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
These bits become "0".

2) "n" bits from the least significant bit become "0".

Cautions

1) Use the function having "_E" in its name to connect a bus.

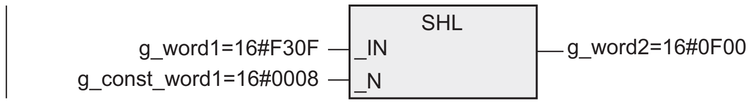
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in s is shifted leftward by "n" bits, and the obtained data is output to a device specified in d using the data type of data stored in a device specified in s.

1) Function without EN/ENO(SHL)

[Structured ladder]

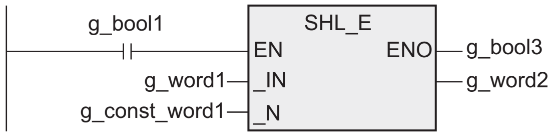


[ST]

```
g_word2:=SHL(g_word1,g_const_word1);
```

2) Function with EN/ENO(SHL_E)

[Structured ladder]



[ST]

```
g_bool3:=SHL_E(g_bool1,g_word1,g_const_word1,g_word2);
```

5.4.2 SHR(_E)

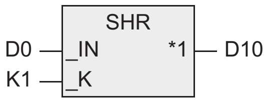
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

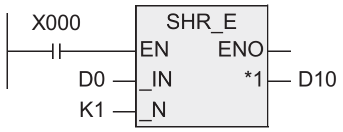
Outline

This function shifts data of specified bit length rightward by the specified number of bits.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

SHR		SHR(_IN,_K);
		Example:
		D10:=
		SHR(D0,K1);

SHR_E	 X000
	SHR_E
	EN
	ENO
	D0
	_IN
	*1
	D10
	K1
	_N
		SHR_E(EN,_IN,_N,Output label);
		Example:
		SHR_E(X000,D0,K1,D10);

*1. Output variable

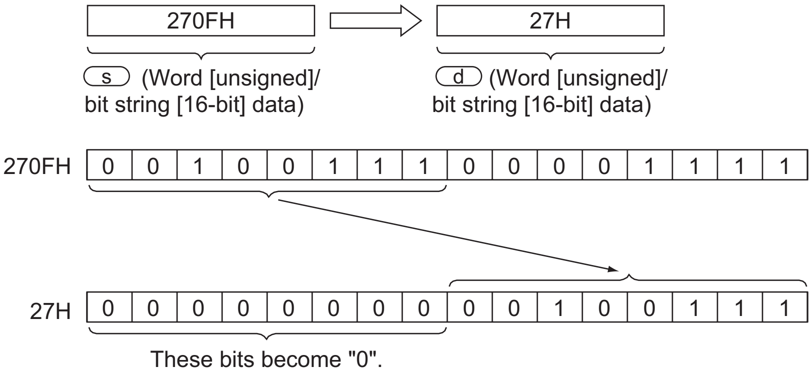
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (ⓢ)	Word device which stores data to be shifted rightward	ANY_BIT
	_K,_N (ⓝ)	Number of shifted bits	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (ⓓ)	Word device which will store data obtained by shift	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function shifts word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in ⓢ rightward by "n" bits, and outputs the obtained data to a device specified in ⓓ using the data type of data stored in a device specified in ⓢ.
Data is shifted rightward by "n" bits specified in ⓝ.
Example: When word [unsigned]/bit string [16-bit] data is stored in a device specified in ⓢ, and "8" is specified in ⓝ



270FH → 27H

ⓢ (Word [unsigned]/ bit string [16-bit] data)   ⓓ (Word [unsigned]/ bit string [16-bit] data)

270FH	0	0	1	0	0	1	1	1	0	0	0	0	1	1	1	1
27H	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1

These bits become "0".

2) "n" bits from the most significant bit become "0".

Cautions

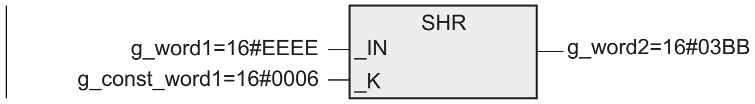
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.

Program example

In this program, word [unsigned]/bit string [16-bit] data stored in a device specified in s is shifted rightward by "n" bits, and the obtained data is output to a device specified in d using the data type of data stored in a device specified in s.

1) Function without EN/ENO(SHR)

[Structured ladder]

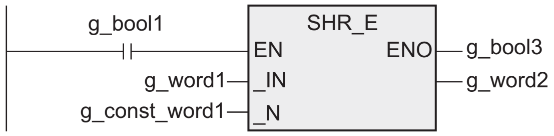


[ST]

```
g_word2:=SHR(g_word1,g_const_word1);
```

2) Function with EN/ENO(SHR_E)

[Structured ladder]



[ST]

```
g_bool3:=SHR_E(g_bool1,g_word1,g_const_word1,g_word2);
```

5.5 Standard Bitwise Boolean Functions

5.5.1 AND_E

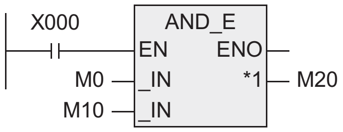
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function obtains the logical product of two or more bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

AND_E		AND_E(EN,_IN,_IN,Output label);
		Example:
		AND_E(X000,M0,M10,M20);

*1. Output variable

2. Variable

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Device used to obtain the logical product	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the logical product using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [unsigned]/bit string [16-bit]

s1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1

Logical product

s2	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	0

⇩

d	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0

2) The number of pins in s can be changed.

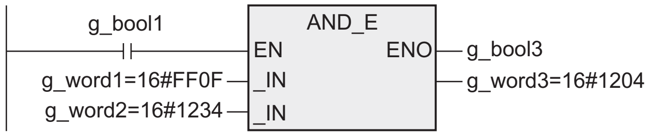
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data. You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the logical product is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

[Structured ladder]



[ST]

```
g_bool3:=AND_E(g_bool1,g_word1,g_word2,g_word3);
```

5.5.2 OR_E

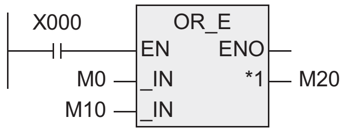
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function obtains the logical sum of two or more bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

OR_E	 X000
	OR_E
	EN
	ENO
	M0
	_IN
	*1
	M20
	M10
	_IN
		OR_E(EN,_IN,_IN,Output label);
		Example:
		OR_E(X000,M0,M10,M20);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Device used to obtain the logical sum	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the logical sum using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [unsigned]/bit string [16-bit]]

s1	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Logical sum																
s2	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0
⇩																
d	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1

2) The number of pins in s can be changed.

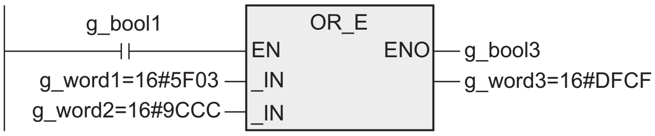
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the logical sum is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in (s1) and (s2), and the operation result is output to a device specified in (d) using the data type of data stored in devices specified in (s1) and (s2).

[Structured ladder]



[ST]

```
g_bool3:=OR_E(g_bool1,g_word1,g_word2,g_word3);
```

5.5.3 XOR_E

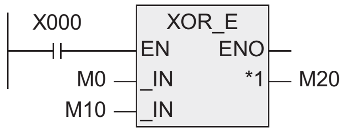
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function obtains the exclusive logical sum of two or more bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

XOR_E	 X000
	XOR_E
	EN
	ENO
	M0
	_IN
	*1
	M20
	M10
	_IN
		XOR_E(EN,_IN,_IN,Output label);
		Example:
		XOR_E(X000,M0,M10,M20);

*1. Output variable

2. .Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 …)	Device used to obtain the exclusive logical sum	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function obtains the exclusive logical sum using each bit of bit/word [unsigned]/bit string [16-bit]/ double word [unsigned]/bit string [32-bit] data stored in devices specified in s1 and s2, and outputs the operation result to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [unsigned]/bit string [16-bit]

s1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Exclusive logical sum

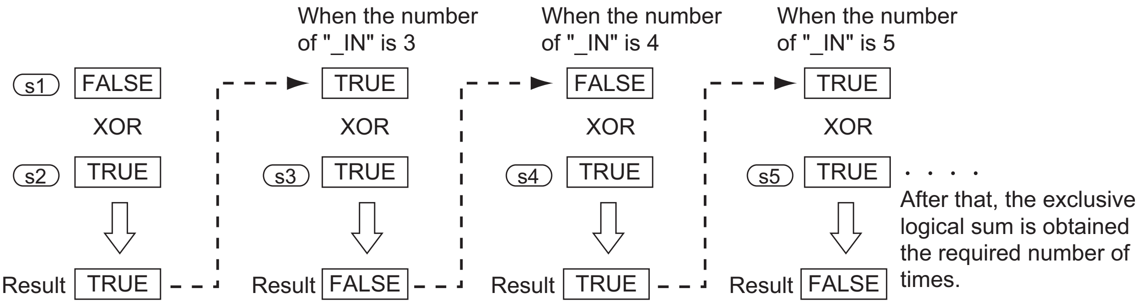
s2	0	0	0	1	1	0	1	1	1	1	1	1	0	0	0	0

⇩

d	1	0	1	1	0	0	0	1	0	1	0	1	1	0	1	0

2) The number of pins in s can be changed.

3) If there are 3 or more s, the exclusive logical sum is obtained using the "exclusive logical sum of s1 and s2" and s3.
If there is s4, the exclusive logical sum is obtained using the "exclusive logical sum of "exclusive logical sum of s1 and s2" and "s3"" and s4. In this way, the exclusive logical sum is obtained the required number of times for all input labels s5 s6 …
Example: When the data type is bit


When the number of "_IN" is 3
When the number of "_IN" is 4
When the number of "_IN" is 5
s1
FALSE
XOR
s2
TRUE
Result
TRUE
TRUE
XOR
s3
TRUE
Result
FALSE
FALSE
XOR
s4
TRUE
Result
TRUE
TRUE
XOR
s5
TRUE
Result
FALSE
. . . .
After that, the exclusive logical sum is obtained the required number of times.

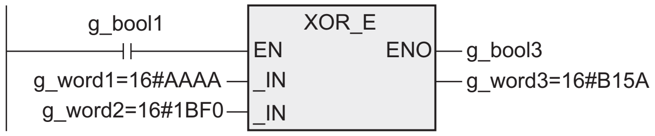
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the exclusive logical sum is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in devices specified in s1 and s2, and the operation result is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

[Structured ladder]



[ST]

```
g_bool3:=XOR_E(g_bool1,g_word1,g_word2,g_word3);
```

5.5.4 NOT(_E)

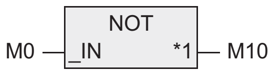
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

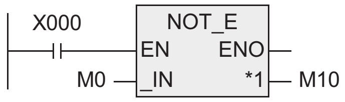
Outline

This function obtains the logical negation of bits, and outputs it.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

NOT		NOT(_IN);
		Example:
		M10:=
		NOT(M0);

NOT_E		NOT_E(EN,_IN,Output label);
		Example:
		NOT_E(X000,M0,M10);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Device used to obtain the logical negation	ANY_BIT
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the operation result	ANY_BIT

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function obtains the logical negation using each bit of bit/word [unsigned]/bit string [16-bit]/double word [unsigned]/bit string [32-bit] data stored in a device specified in s, and outputs the operation result to a device specified in d using the data type of data stored in a device specified in s.
Example: When the data type is word [unsigned]/bit string [16-bit]

s	0	1	1	0	1	0	1	1	0	0	0	0	1	1	1	1

Logical negation

d	1	0	0	1	0	1	0	0	1	1	1	1	0	0	0	0

Cautions

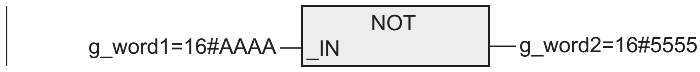
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the logical negation is obtained using each bit of word [unsigned]/bit string [16-bit] data stored in a device specified in s, and the operation result is output to a device specified in d using the data type of data stored in a device specified in s.

1) Function without EN/ENO(NOT)

[Structured ladder]

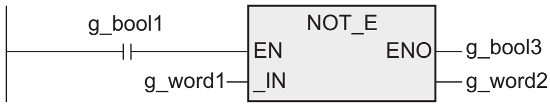


[ST]

```
g_word2:= NOT(g_word1);
```

2) Function with EN/ENO(NOT_E)

[Structured ladder]



[ST]

```
g_bool3:=NOT_E(g_bool1,g_word1,g_word2);
```

5.6 Standard Selection Functions

5.6.1 SEL(_E)

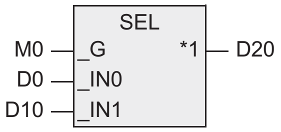
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

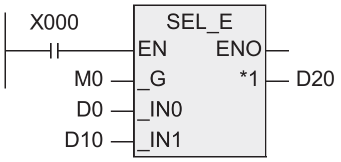
Outline

This function selects either one between two data in accordance with the input condition, and outputs the selection result.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

SEL		SEL(_G,_IN0,_IN1);
		Example:
		D20:=
		SEL(M0,D0,D10);

SEL_E	 X000
	SEL_E
	EN
	ENO
	M0
	_G
	*1
	D20
	D0
	_IN0
	D10
	_IN1
		SEL_E(EN,_G,_IN0,_IN1,Output label);
		Example:
		SEL_E(X000,M0,D0,D10,D20);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_G (s1)	Bit data used as the selection condition	Bit
	_IN0 (s2)	Selectable data, or word device which stores such data	ANY
	_IN1 (s3)	Selectable data, or word device which stores such data	ANY
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the selection result	ANY

In explanation of functions, I/O variables inside ( ) are described.

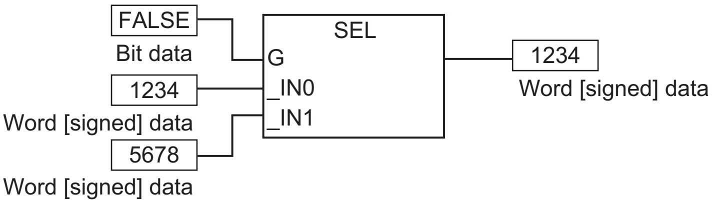
Explanation of function and operation

This function outputs either one between the values stored in devices specified in s2 and s3 in accordance with the value stored in a device specified in s1 to a device specified in d using the data type of data stored in a device specified in s2 and s3.

When the value stored in a device specified in s1 is "FALSE", this function outputs the value stored in a device specified in s2 to a device specified in d.

When the value stored in a device specified in s1 is "TRUE", this function outputs the value stored in a device specified in s3 to a device specified in d.

Example: When the data type of input variables s2 and s3 is word [signed]



Cautions

1) Use the function having "_E" in its name to connect a bus.

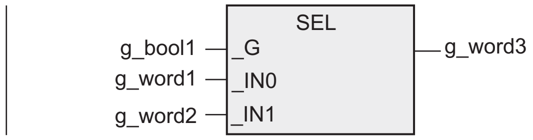
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, either one between the values stored in devices specified in s2 and s3 is output in accordance with the value stored in a device specified in s1 to a device specified in d using the data type of data stored in devices specified in s2 and s3.

1) Function without EN/ENO(SEL)

[Structured ladder]

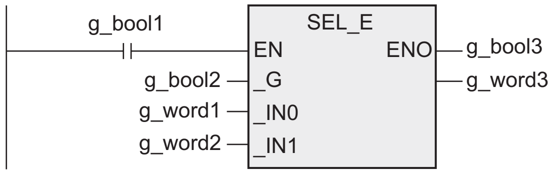


[ST]

```
g_word3:=SEL(g_bool1,g_word1,g_word2);
```

2) Function with EN/ENO(SEL_E)

[Structured ladder]



[ST]

```
g_bool3:=SEL_E(g_bool1,g_bool2,g_word1,g_word2,g_word3);
```

5.6.2 MAXIMUM(_E)

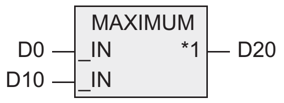
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

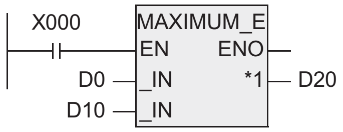
Outline

This function searches the maximum value among data, and outputs the maximum value.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MAXIMUM		MAXIMUM(_IN,_IN);
		Example:
		D20:=
		MAXIMUM(D0,D10);

MAXIMUM_E	 X000
	MAXIMUM_E
	EN
	ENO
	D0
	D10
	_IN
	_IN
	*1
	D20
		MAXIMUM_E(EN,_IN,_IN,Output label);
		Example:
		MAXIMUM_E(X000,D0,D10,D20);

*1. Output variable

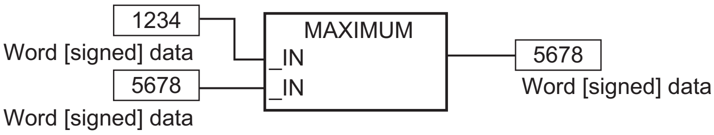
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the maximum value	ANY_SIMPLE

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function outputs the maximum value among ANY_SIMPLE type data stored in devices specified in s1 and s2 to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]



2) The number of pins in s can be changed.

Cautions

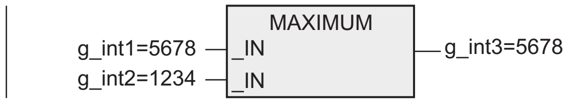
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the maximum value among word [signed] data stored in devices specified in s1 and s2 is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

1) Function without EN/ENO(MAXIMUM)

[Structured ladder]

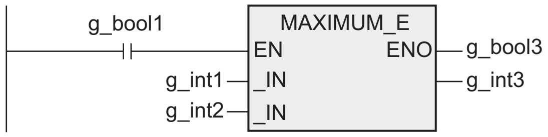


[ST]

```
g_int3:=MAXIMUM(g_int1,g_int2);
```

2) Function with EN/ENO(MAXIMUM_E)

[Structured ladder]



[ST]

```
g_bool3:=MAXIMUM_E(g_bool1,g_int1,g_int2,g_int3);
```

5.6.3 MINIMUM(_E)

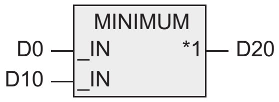
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

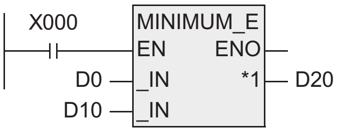
Outline

This function searches the minimum value among data, and outputs the minimum value.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MINIMUM		MINIMUM(_IN,_IN);
		Example:
		D20:=
		MINIMUM(D0,D10);

MINIMUM_E	 X000
	MINIMUM_E
	EN
	ENO
	D0
	D10
	_IN
	_IN
	*1
	D20
		MINIMUM_E(EN,_IN,_IN,Output label);
		Example:
		MINIMUM_E(X000,D0,D10,D20);

*1. Output variable

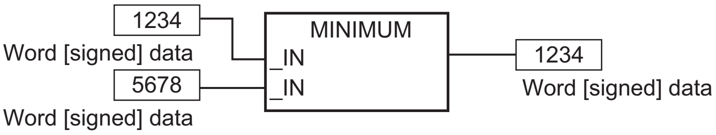
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the minimum value	ANY_SIMPLE

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function outputs the minimum value among ANY_SIMPLE type data stored in devices specified in s1 and s2 to a device specified in d using the data type of data stored in devices specified in s1 and s2.
Example: When the data type is word [signed]



2) The number of pins in s can be changed.

Cautions

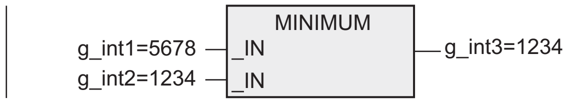
1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the minimum value among word [signed] data stored in devices specified in s1 and s2 is output to a device specified in d using the data type of data stored in devices specified in s1 and s2.

1) Function without EN/ENO(MINIMUM)

[Structured ladder]

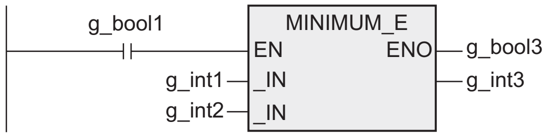


[ST]

```
g_int3:=MINIMUM(g_int1,g_int2);
```

2) Function with EN/ENO(MINIMUM_E)

[Structured ladder]



[ST]

```
g_bool3:=MINIMUM_E(g_bool1,g_int1,g_int2,g_int3);
```

5.6.4 LIMITATION(_E)

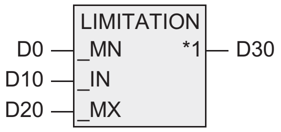
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

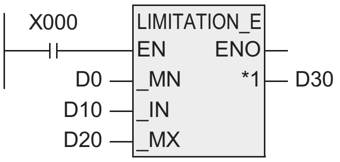
Outline

This function judges whether data is located within the range between the upper limit value and the lower limit value.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

LIMITATION	 LIMITATION
	D0
	_MN
	*1
	D30
	D10
	_IN
	D20
	_MX
		LIMITATION(_MN,_IN,_MX);
		Example:
		D30:=
		LIMITATION(D0,D10,D20);

LIMITATION_E	 X000
	LIMITATION_E
	EN
	ENO
	D0
	_MN
	*1
	D30
	D10
	_IN
	D20
	_MX
		LIMITATION_E(EN,_MN,_IN,_MX, Output label);
		Example:
		LIMITATION_E(X000,D0,D10,D20, D30);

*1. Output variable

2. Set data

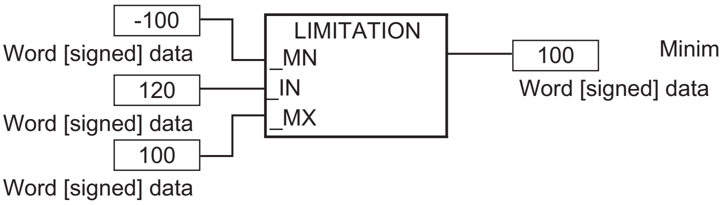
Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_MN (s1)	Lower limit data, or word device which stores such data	ANY_SIMPLE
	_IN (s2)	Input data, or word device which stores such data	ANY_SIMPLE
	_MX (s3)	Upper limit data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Word device which will store the output data	ANY_SIMPLE

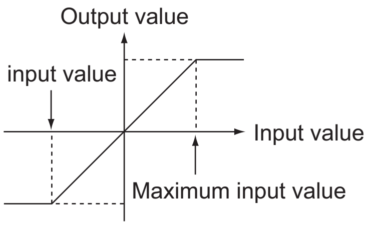
In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function outputs data whose type is same as the data stored in devices specified in s1, s2 and s3 to a device specified in d in accordance with ANY_SIMPLE type data stored in devices specified in s1, s2 and s3.

1) In the case of "Contents of a device specified in s2 > Contents of a device specified in s3", this function outputs the contents of a device specified in s3 to a device specified in d.
2) In the case of "Contents of a device specified in s2 < Contents of a device specified in s1", this function outputs the contents of a device specified in s1 to a device specified in d.
3) In the case of "Contents of a device specified in s1 ≤ Contents of a device specified in s2 ≤ Contents of a device specified in s3", this function outputs the contents of a device specified in s2 to a device specified in d.
Example: When the data type is word [signed]





Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

An operation error occurs when this function is executed in the following setting status. The error flag M8067 turns ON, and D8067 stores the error code K6706.

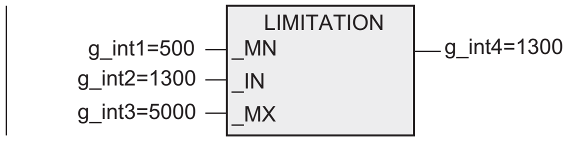
Contents of a device specified in s1 > Contents of a device specified in s3
(Lower limit data) (Upper limit data)

Program example

In this program, data whose type is same as the data stored in devices specified in s1, s2 and s3 is output to a device specified in d in accordance with ANY_SIMPLE type data stored in devices specified in s1, s2 and s3.

1) Function without EN/ENO(LIMITATION)

[Structured ladder]

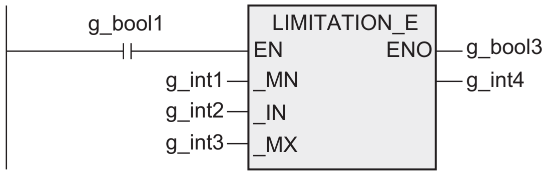


[ST]

```
g_int4:=LIMITATION(g_int1,g_int2,g_int3);
```

2) Function with EN/ENO(LIMITATION_E)

[Structured ladder]



[ST]

```
g_bool3:=LIMITATION_E(g_bool1,g_int1,g_int2,g_int3,g_int4);
```

5.6.5 MUX(_E)

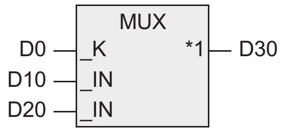
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

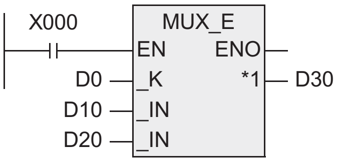
Outline

This function selects data, and outputs the selected data.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MUX		MUX(_K,_IN,_IN);
		Example:
		D30:=
		MUX(D0,D10,D20);

MUX_E	 X000
	MUX_E
	EN
	ENO
	D0
	D10
	D20
	_K
	_IN
	_IN
	*1
	D30
		MUX_E(EN,_K,_IN,_IN,Output label);
		Example:
		MUX_E(X000,D0,D10,D20,D30);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_K (n)	Selection data, or word device which stores such data	Word [signed]
	_IN (s1 …)	Selectable data, or word device which stores such data	ANY
Output variable	ENO	Execution status	Bit
	*1	Word device which will store the selected data	ANY

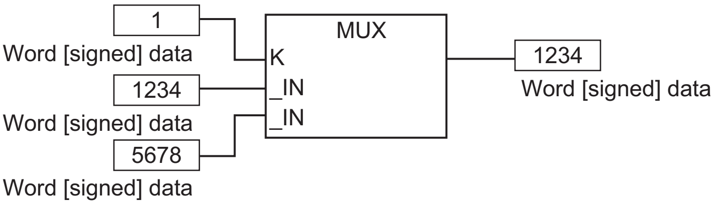
In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function outputs either one among values stored in devices specified in s1 … to a device specified in d in accordance with the value specified in n using the data type of data stored in devices specified in s1 ….

a) When the value specified in n is "1", this function outputs the value stored in a device specified in s1 to a device specified in d.

b) When the value specified in n is "n", this function outputs the value stored in a device specified in sn to a device specified in d.
Example: When the data type is word [signed]



2) When a value input to n is outside the pin number range for s1 …, this function outputs an indefinite value to a device specified in d.
(An operation error does not occur. "MUX_E" outputs "FALSE" from ENO.)

3) The number of pins in s can be changed.

Cautions

1) Use the function having "_E" in its name to connect a bus.

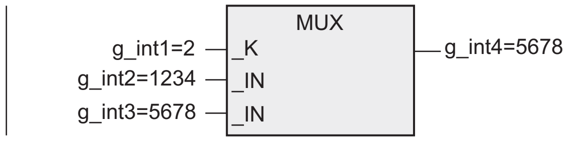
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this example, either one among values stored in devices specified in (s1) … is output to a device specified in (d) in accordance with the value specified in (n) using the data type of data stored in devices specified in (s1) ….

1) Function without EN/ENO(MUL)

[Structured ladder]

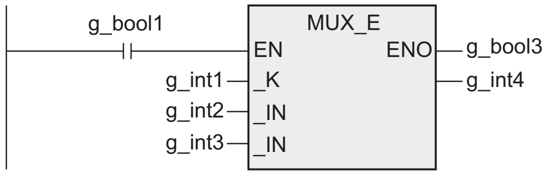


[ST]

```
g_int4:=MUX(g_int1,g_int2,g_int3);
```

2) Function with EN/ENO(MUL_E)

[Structured ladder]



[ST]

```
g_bool3:=MUX_E(g_bool1,g_int1,g_int2,g_int3,g_int4);
```

1 Outline
2 Function List
3 Function Construction
4 How to Read Explanation of Functions
5 Applied Functions
6 Standard Function Blocks
A Correspondence between Devices and Addresses

5.7 Standard Comparison Functions

5.7.1 GT_E

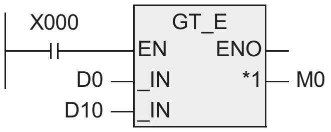
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "> (larger)".

1. Format

Function name	Expression in each language	
	Structured ladder	ST

GT_E	 X000
	GT_E
	EN
	ENO
	D0
	_IN
	*1
	M0
	D10
	_IN
		GT_E(EN,_IN,_IN,Output label);
		Example:
		GT_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in s1 ..., and outputs the operation result expressed as the bit type data to a device specified in d.
   This function executes comparison [s1 > s2] & [s2 > s3] & ... & [sn-1 > sn].
   a) This function outputs "TRUE" when all comparison results are "s(n-1) > s(n)".
   b) This function outputs "FALSE" when any comparison result is "s(n-1) ≤ s(n)".
2) The number of pins in s can be changed.

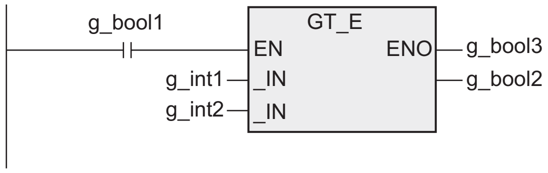
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects.
Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=GT_E(g_bool1,g_int1,g_int2,g_bool2);
```

5.7.2 GE_E

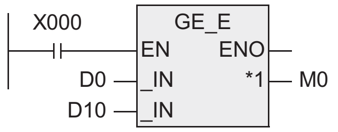
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≥ (larger or equal)".

1. Format

Function name	Expression in each language	
	Structured ladder	ST

GE_E	 X000
	GE_E
	EN
	ENO
	D0
	_IN
	*1
	M0
	D10
	_IN
		GE_E(EN,_IN,_IN,Output label);
		Example:
		GE_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 …)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in s1 …, and outputs the operation result expressed as the bit type data to a device specified in d.
   This function executes comparison [s1 ≥ s2] & [s2 ≥ s3] & … & [sn-1 ≥ sn].
   a) This function outputs "TRUE" when all comparison results are "s(n-1) ≥ s(n)".
   b) This function outputs "FALSE" when any comparison result is "s(n-1) < s(n)".
2) The number of pins in s can be changed.

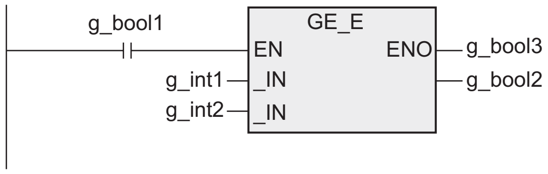
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=GE_E(g_bool1,g_int1,g_int2,g_bool2);
```

5.7.3 EQ_E

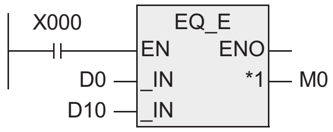
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "= (equal)".

1. Format

Function name	Expression in each language	
	Structured ladder	ST

EQ_E	 X000
	EQ_E
	EN
	ENO
	D0
	_IN
	*1
	M0
	D10
	_IN
		EQ_E(EN,_IN,_IN,Output label);
		Example:
		EQ_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 ...)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in s1 ..., and outputs the operation result expressed as the bit type data to a device specified in d.
This function executes comparison [s1 = s2] & [s2 = s3] & ... & [sn-1 = sn].
a) This function outputs "TRUE" when all comparison results are "s(n-1) = s(n)".
b) This function outputs "FALSE" when any comparison result is "s(n-1) ≠ s(n)".

2) The number of pins in s can be changed.

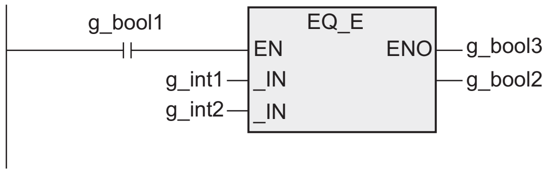
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=EQ_E(g_bool1,g_int1,g_int2,g_bool2);
```

5.7.4 LE_E

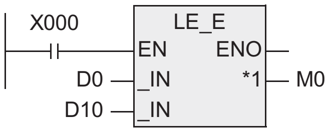
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≤ (smaller or equal)".

1. Format

Function name	Expression in each language	
	Structured ladder	ST

LE_E	 X000
	LE_E
	EN
	ENO
	D0
	_IN
	*1
	M0
	D10
	_IN
		LE_E(EN,_IN,_IN,Output label);
		Example:
		LE_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 …)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in s1 …, and outputs the operation result expressed as the bit type data to a device specified in d.
   This function executes comparison [s1 ≤ s2] & [s2 ≤ s3] & … & [sn-1 ≤ sn].
   a) This function outputs "TRUE" when all comparison results are "s(n-1) ≤ s(n)".
   b) This function outputs "FALSE" when any comparison result is "s(n-1) > s(n)".
2) The number of pins in s can be changed.

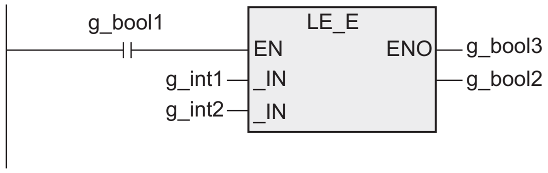
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the contents of devices specified in (s1) and (s2) are compared, and the operation result is output to a device specified in (d).

[Structured ladder]



[ST]

```
g_bool3:=LE_E(g_bool1,g_int1,g_int2,g_bool2);
```

5.7.5 LT_E

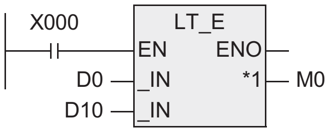
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "< (smaller)".

1. Format

Function name	Expression in each language	
	Structured ladder	ST

LT_E	 X000
	LT_E
	EN
	ENO
	D0
	_IN
	*1
	M0
	D10
	_IN
		LT_E(EN,_IN,_IN,Output label);
		Example:
		LT_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1 …)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function compares the contents of devices specified in s1 …, and outputs the operation result expressed as the bit type data to a device specified in d.
This function executes comparison [s1 < s2] & [s2 < s3] & … & [sn-1 < sn].
   a) This function outputs "TRUE" when all comparison results are "s(n-1) < s(n)".
   b) This function outputs "FALSE" when any comparison result is "s(n-1) ≥ s(n)".
2) The number of pins in s can be changed.

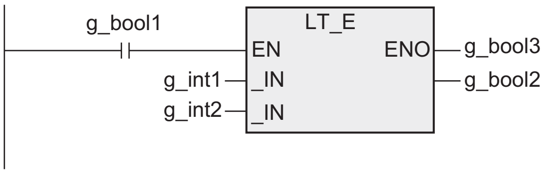
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Program example

In this program, the contents of devices specified in s1 and s2 are compared, and the operation result is output to a device specified in d.

[Structured ladder]



[ST]

```
g_bool3:=LT_E(g_bool1,g_int1,g_int2,g_bool2);
```

5.7.6 NE_E

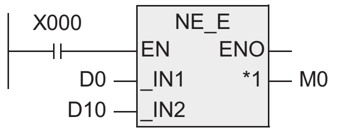
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function compares data with regard to "≠ (unequal)".

1. Format

Function name	Expression in each language	
	Structured ladder	ST

NE_E	 X000
	NE_E
	EN
	ENO
	D0
	_IN1
	*1
	M0
	D10
	_IN2
		NE_E(EN,_IN1,_IN2,Output label);
		Example:
		NE_E(X000,D0,D10,M0);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Compared data, or word device which stores such data	ANY_SIMPLE
	_IN2 (s2)	Compared data, or word device which stores such data	ANY_SIMPLE
Output variable	ENO	Execution status	Bit
	*1 (d)	Device which will store the comparison result	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function compares the contents of devices specified in s1 and s2, and outputs the operation result expressed as the bit type data to a device specified in d.

This function executes comparison [s1≠s2].

a) This function outputs "TRUE" when in the case of "s1≠s2"

b) This function outputs "FALSE" when in the case of "s1=s2"

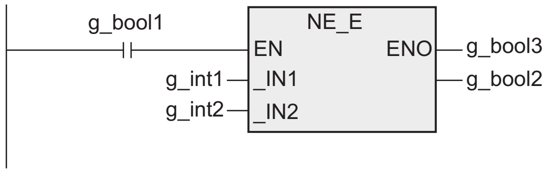
Cautions

When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices. Use global labels when specifying labels.

Program example

In this program, the contents of devices specified in (s1) and (s2) are compared, and the operation result is output to a device specified in (d).

[Structured ladder]



[ST]

```
g_bool3:=NE_E(g_bool1,g_int1,g_int2,g_bool2);
```

5.8 Standard Character String Functions

5.8.1 MID(_E)

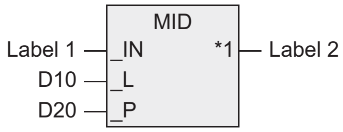
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

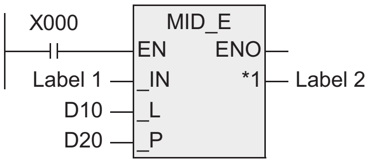
Outline

This function obtains a character string from a specified position.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MID	 MID
	Label 1
	D10
	D20
	_IN
	_L
	_P
	*1
	Label 2
		MID(_IN,_L ,_P);
		Example:
		Label 2:=
		MID(Label 1,D10,D20);

MID_E	 X000
	MID_E
	EN
	ENO
	Label 1
	D10
	D20
	_IN
	_L
	_P
	*1
	Label 2
		MID_E(EN,_IN,_L ,_P,Output label);
		Example:
		MID_E(X000,Label 1,D10,D20,
		Label 2);

*1. Output variable

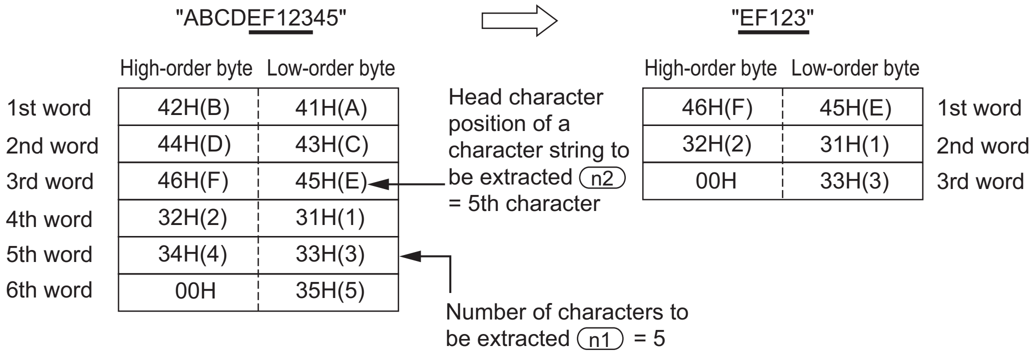
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Head word device which stores a character string	String
	_L (n1)	Word device which stores the number of characters to be obtained	Word [signed]
	_P (n2)	Word device which stores the head character position of a character string to be obtained	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the obtained character string	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function extracts specified number of characters from an arbitrary position of a character string stored in devices specified in s, and outputs the obtained data to devices specified in d. The value specified in n1 specifies the number of characters to be extracted.
The value specified in n2 specifies the head character position of characters to be extracted.
Example: When "5" is specified in n1 and n2



"ABCDEF12345" ⟹ "EF123"

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	46H(F)	45H(E)
4th word	32H(2)	31H(1)
5th word	34H(4)	33H(3)
6th word	00H	35H(5)

Head character position of a character string to be extracted n2 = 5th character

Number of characters to be extracted n1 = 5

High-order byte	Low-order byte	
46H(F)	45H(E)	1st word
32H(2)	31H(1)	2nd word
00H	33H(3)	3rd word

2) A character string (data) stored in devices specified in s indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.
3) When the number of characters to be extracted specified in n1 is "0", this function does not execute processing.
4) When the number of characters to be extracted specified in n1 is "-1", this function outputs the final character of a character string specified in s to devices specified in d.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

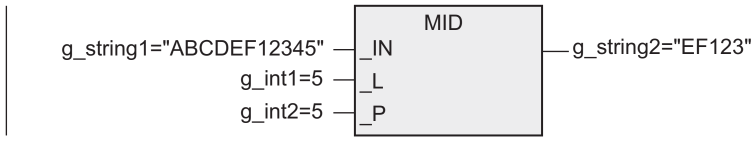
1) When "00H" is not set in the corresponding device range after the device specified in s
(Error code: K6706)
2) When the head character position specified in n2 exceeds the number of characters of a character string stored in devices specified in s
(Error code: K6706)
3) When the number of characters specified in n1 exceeds the range of devices specified in d
(Error code: K6706)
4) When the number of devices after the device number specified in d is smaller than the number of devices required for storing an extracted character string
(In this case, "00H" cannot be stored after all character strings and the final character.)
(Error code: K6706)
5) When the value specified in n2 is negative
(Error code: K6706)
6) When the value specified in n1 is "-2" or less
(Error code: K6706)
7) When the value specified in n1 exceeds the number of characters of a character string stored in devices specified in s
(Error code: K6706)

Program example

In this program, specified number of characters are extracted from an arbitrary position of a character string stored in devices specified in s, and the obtained data is output to devices specified in d.

1) Function without EN/ENO(MID)

[Structured ladder]

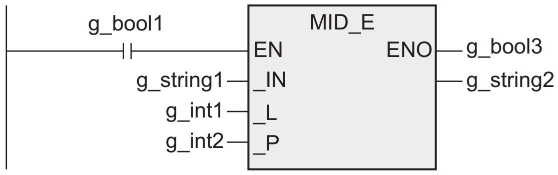


[ST]

```
g_string2:=MID(g_string1,g_int1,g_int2);
```

2) Function with EN/ENO(MID_E)

[Structured ladder]



[ST]

```
g_bool3:=MID_E(g_bool1,g_string1,g_int1,g_int2,g_string2);
```

5.8.2 CONCAT(_E)

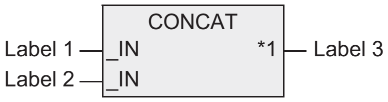
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

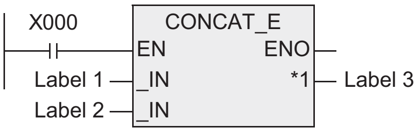
Outline

This function connects character strings.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

CONCAT		CONCAT(_IN,_IN);
		Example:
		Label 3:=
		CONCAT(Label 1,Label 2);

CONCAT_E	 X000
	CONCAT_E
	EN
	ENO
	Label 1
	_IN
	*1
	Label 3
	Label 2
	_IN
		CONCAT_E(EN,_IN,_IN,Output label);
		Example:
		CONCAT_E(X000,Label 1,Label 2, Label 3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s1) _IN (s2)	Head word device which stores the data (character string) to be connected, or directly specified character string	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the connected data (character string)	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function connects a character string stored in devices specified in s2 after a character string stored in devices specified in s1, and outputs the character string obtained by connection to devices specified in d.
When connecting a character string stored in devices specified in s2, this function ignores "00H" which indicates the end of a character string stored in devices specified in s1.
After two character strings are connected, "00H" is automatically added at the end.

"ABCDE" + "123456" ⇨ "ABCDE123456"

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	00H	45H(E)

	High-order byte	Low-order byte
1st word	32H(2)	31H(1)
2nd word	34H(4)	33H(3)
3rd word	36H(6)	35H(5)
4th word	0000H	

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	31H(1)	45H(E)
4th word	33H(3)	32H(2)
5th word	35H(5)	34H(4)
6th word	00H	36H(6)

2) A character string (data) stored in devices specified in s indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) For direct specification, up to 32 characters can be specified (input).
When word devices are specified in s1 and s2, this restriction (up to 32 characters) is not applicable.

4) When both a character string stored in devices specified in s1 and a character string stored in devices specified in s2 begin with "00H" (when character = 0), this function stores "0000H" in devices specified in d.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When the number of devices after the device number specified in d is smaller than the number of devices required for storing the character string obtained by connection
(In this case, "00H" cannot be stored after all character strings and final character.)
(Error code: K6706)

2) When devices which store character strings specified in s1 and s2 overlap device numbers specified in d which will store the character string obtained by connection
(Error code: K6706)

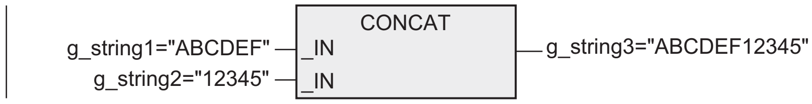
3) When "00H" does not exist in the corresponding device range after devices specified in s1 and s2
(Error code: K6706)

Program example

In this program, a character string stored in devices specified in s2 is connected after a character string stored in devices specified in s1, and the character string obtained by connection is output to devices specified in d.

1) Function without EN/ENO(CONCAT)

[Structured ladder]

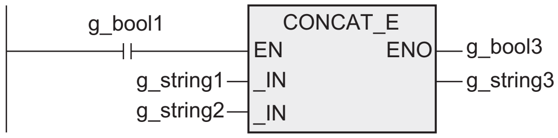


[ST]

```
g_string3:=CONCAT(g_string1,g_string2);
```

2) Function with EN/ENO(CONCAT_E)

[Structured ladder]



[ST]

```
g_bool3:=CONCAT_E(g_bool1,g_string1,g_string2,g_string3);
```

5.8.3 INSERT(_E)

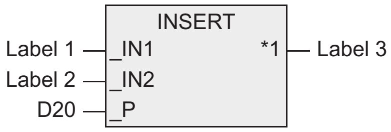
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

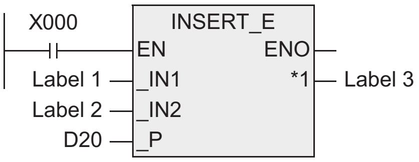
Outline

This function inserts a character string.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

INSERT	 INSERT
	Label 1
	_IN1
	Label 2
	_IN2
	D20
	_P
	*1
	Label 3
		INSERT(_IN1,_IN2,_P);
		Example:
		Label 3:=
		INSERT(Label 1,Label 2,D20);

INSERT_E	 X000 INSERT_E EN ENO Label 1 _IN1 *1 Label 3 Label 2 _IN2 D20 _P	INSERT_E(EN,_IN1,_IN2,_P, Output label); Example: INSERT_E(X000,Label 1,Label 2, D20,Label 3);

*1. Output variable

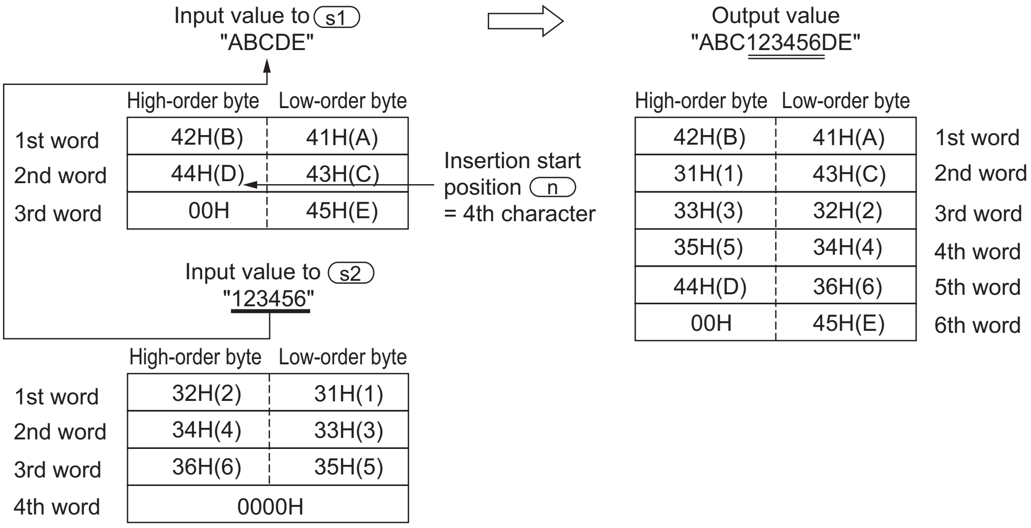
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores a character string to get insertion	String
	_IN2 (s2)	Head word device which stores a character string to be inserted	String
	_P (n)	Word device which stores a character position to get insertion	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store a character string obtained by insertion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function inserts a character string stored in devices specified in s2 into an arbitrary position (counted from the head) of a character string stored in devices specified in s1, and outputs the character string obtained by insertion to devices specified in d.
The value specified in n specifies the position from which the character string stored in devices specified in s2 is inserted.
After inserting a character string stored in devices specified in s2 into a character string stored in devices specified in s1, this function ignores "00H" which indicates the end of a character string stored in devices specified in s2.
Example: When "4" is specified in n



Input value to s1 "ABCDE"

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	00H	45H(E)

Input value to s2 "123456"

	High-order byte	Low-order byte
1st word	32H(2)	31H(1)
2nd word	34H(4)	33H(3)
3rd word	36H(6)	35H(5)
4th word	0000H	

Output value "ABC123456DE"

High-order byte	Low-order byte	
42H(B)	41H(A)	1st word
31H(1)	43H(C)	2nd word
33H(3)	32H(2)	3rd word
35H(5)	34H(4)	4th word
44H(D)	36H(6)	5th word
00H	45H(E)	6th word

2) A character string (data) stored in devices specified in s indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

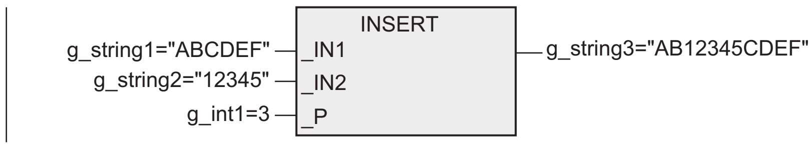
1) When the number of devices after the device number specified in d is smaller than the number of devices required for storing the output data obtained by insertion
(Error code: K6706)
2) When devices which store character strings specified in s1 and s2 overlap device numbers specified in d which will store the character string obtained by connection
(Error code: K6706)
3) When "00H" does not exist in the corresponding device range after devices specified in s1 and s2
(Error code: K6706)
4) When the number of characters of a character string stored in devices specified in s2 is 32768 or more
(Error code: K6706)
5) When the value specified in n is negative
(Error code: K6706)

Program example

In this program, a character string stored in devices specified in s2 is inserted into an arbitrary position (counted from the head) of a character string stored in devices specified in s1, and the character string obtained by insertion is output to devices specified in d.

1) Function without EN/ENO(INSERT)

[Structured ladder]

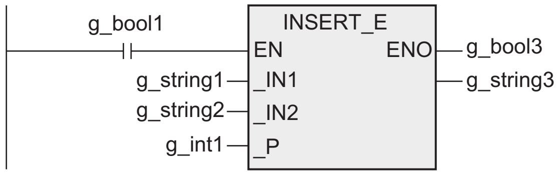


[ST]

```
g_string3:=INSERT(g_string1,g_string2,g_int1);
```

2) Function with EN/ENO(INSERT_E)

[Structured ladder]



[ST]

```
g_bool3:=INSERT_E(g_bool1,g_string1,g_string2,g_int1,g_string3);
```

5.8.4 DELETE(_E)

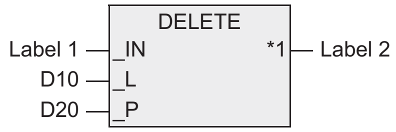
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

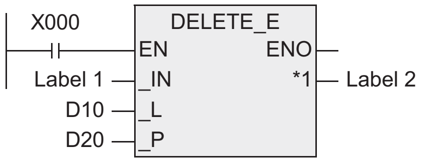
Outline

This function deletes a character string.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DELETE	 DELETE
	Label 1
	D10
	D20
	_IN
	_L
	_P
	*1
	Label 2
		DELETE(_IN,_L ,_P);
		Example:
		Label 2:=
		DELETE(Label 1,D10,D20);

DELETE_E	 X000 DELETE_E EN ENO Label 1 D10 D20 _IN _L _P *1 Label 2	DELETE_E(EN,_IN,_L ,_P, Output label); Example: DELETE_E(X000, Label 1, D10, D20, Label 2);

*1. Output variable

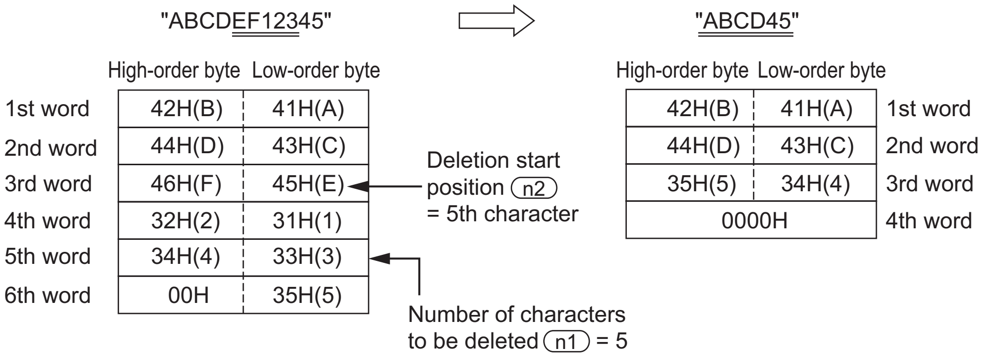
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN (s)	Head word device which stores a character string to get deletion	String
	_L (n1)	Number of characters to be deleted	Word [signed]
	_P (n2)	Head position to get deletion	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store a character string remaining after deletion	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function deletes specified number of characters from an arbitrary position of a character string stored in devices specified in (s), and outputs the character string remaining after deletion to devices specified in (d).
The value specified in (n1) specifies the number of characters to be deleted.
The value specified in (n2) specifies the position from which specified number of characters are deleted.
Example: When "5" is specified in (n1) and (n2)



2) A character string (data) stored in devices specified in (s) indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H" does not exist in the corresponding device range after the device specified in s
(Error code: K6706)

2) When the number of characters of a character string stored in devices specified in s is 32768 or more
(Error code: K6706)

3) When the number of devices after the device number specified in d is smaller than the number of devices required for storing the character string remaining after deletion of specified number of characters
(Error code: K6706)

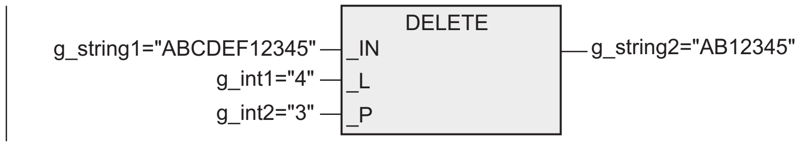
4) When the value specified in n2 is negative
(Error code: K6706)

Program example

In this program, specified number of characters are deleted from an arbitrary position of a character string stored in devices specified in s, and the character string remaining after deletion is output to devices specified in d.

1) Function without EN/ENO(DELETE)

[Structured ladder]

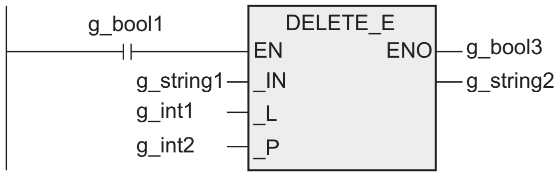


[ST]

```
g_string2:=DELETE(g_string1,g_int1,g_int2);
```

2) Function with EN/ENO(DELETE_E)

[Structured ladder]



[ST]

```
g_bool3:=DELETE_E(g_bool1,g_string1,g_int1,g_int2,g_string2);
```

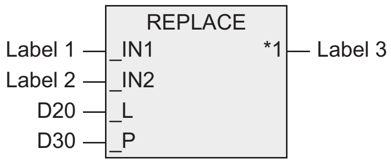
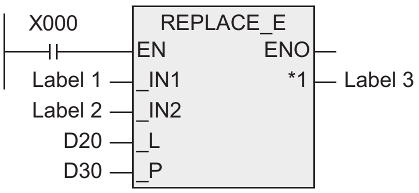
5.8.5 REPLACE(_E)

FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

Outline

This function replaces a character string.

1. Format

Function name	Expression in each language	
	Structured ladder	ST
REPLACE	 REPLACE Label 1 _IN1 *1 Label 3 Label 2 _IN2 D20 _L D30 _P	REPLACE(_IN1,_IN2,_L ,_P); Example: Label 3:= REPLACE(Label 1,Label 2, D20,D30);
REPLACE_E	 X000 REPLACE_E EN ENO Label 1 _IN1 *1 Label 3 Label 2 _IN2 D20 _L D30 _P	REPLACE_E(EN,_IN1,_IN2, _L ,_P,Output label); Example: REPLACE_E(X000,Label 1, Label 2,D20,D30,Label 3);

*1. Output variable

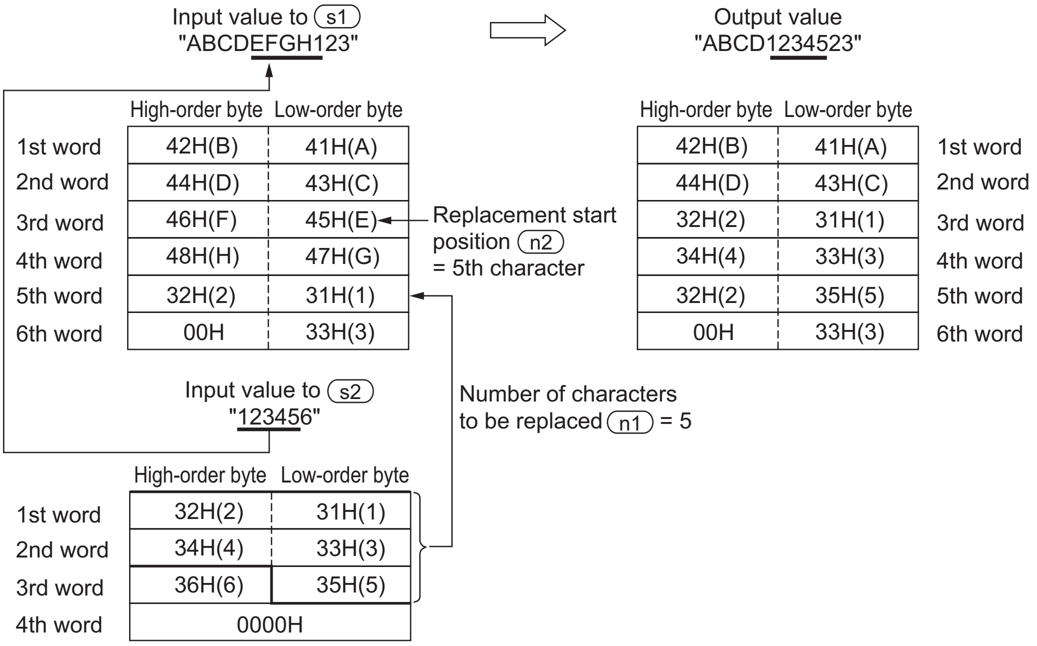
2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores a character string to be replaced	String
	_IN2 (s2)	Head word device which stores a replacement character string	String
	_L (n1)	Word device which stores the number of characters to be replaced	Word [signed]
	_P (n2)	Word device which stores the head character position to be replaced in a character string to be replaced	Word [signed]
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store a character string obtained by replacement	String

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function replaces specified number of characters from an arbitrary position of a character string stored in devices specified in s1 with a character string stored in devices specified in s2, and outputs the character string obtained by replacement to devices specified in d.
The value specified in n1 specifies the number of characters to be replaced.
The value specified in n2 specifies the position from which specified number of characters are replaced.
Example: When "5" is specified in n1 and n2


Input value to s1
"ABCDEFGH123"
Output value
"ABCD1234523"
Replacement start position n2 = 5th character
Input value to s2
"123456"
Number of characters to be replaced n1 = 5

	High-order byte	Low-order byte
1st word	42H(B)	41H(A)
2nd word	44H(D)	43H(C)
3rd word	46H(F)	45H(E)
4th word	48H(H)	47H(G)
5th word	32H(2)	31H(1)
6th word	00H	33H(3)

High-order byte	Low-order byte	
42H(B)	41H(A)	1st word
44H(D)	43H(C)	2nd word
32H(2)	31H(1)	3rd word
34H(4)	33H(3)	4th word
32H(2)	35H(5)	5th word
00H	33H(3)	6th word

	High-order byte	Low-order byte
1st word	32H(2)	31H(1)
2nd word	34H(4)	33H(3)
3rd word	36H(6)	35H(5)
4th word	0000H	

2) A character string (data) stored in devices specified in s indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) When "n1+n2" exceeds the number of characters of a character string stored in devices specified in s1, excessive characters are not output to devices specified in d.

4) When "-1" is specified in n1, the number of characters of a character string stored in devices specified in s2 is regarded as the value specified in n1.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H" does not exist in the corresponding device range after the devices specified in (s1) and (s2)
(Error code: K6706)

2) When the value specified in (n1) exceeds the number of characters of a character string stored in devices specified in (s2)
(Error code: K6706)

3) When the value specified in (n2) is negative
(Error code: K6706)

4) When the value specified in (n1) is "-2" or less
(Error code: K6706)

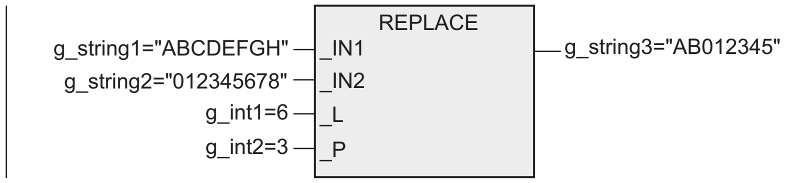
5) When the value specified in (n2) exceeds the number of characters of a character string stored in devices specified in (s1)
(Error code: K6706)

Program example

In this program, specified number of characters starting from an arbitrary position of a character string stored in devices specified in (s1) are replaced with a character string stored in devices specified in (s2), and the character string obtained by replacement is output to devices specified in (d).

1) Function without EN/ENO(REPLACE)

[Structured ladder]

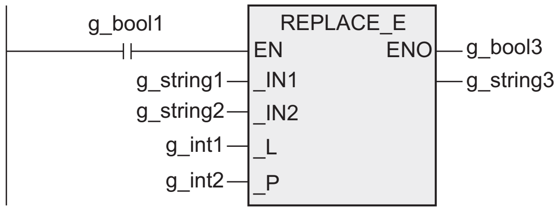


[ST]

```
g_string3:=REPLACE(g_string1,g_string2,g_int1,g_int2);
```

2) Function with EN/ENO(REPLACE_E)

[Structured ladder]



[ST]

```
g_bool3:=REPLACE_E(g_bool1,g_string1,g_string2,g_int1,g_int2,g_string3);
```

5.8.6 FIND(_E)

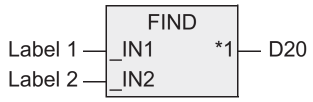
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	×	×	×	×	×	×	×

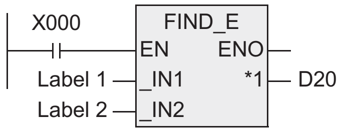
Outline

This function searches a character string.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

FIND		FIND(_IN1,_IN2);
		Example:
		D20:=
		FIND(Label 1,Label 2);

FIND_E	 X000
	FIND_E
	EN
	ENO
	Label 1
	_IN1
	*1
	D20
	Label 2
	_IN2
		FIND_E(EN,_IN1,_IN2,Output label);
		Example:
		FIND_E(X000,Label 1,Label 2, D20);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores a character string to get search	String
	_IN2 (s2)	Head word device which stores a character string to be searched	String
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the search result	Word [signed]

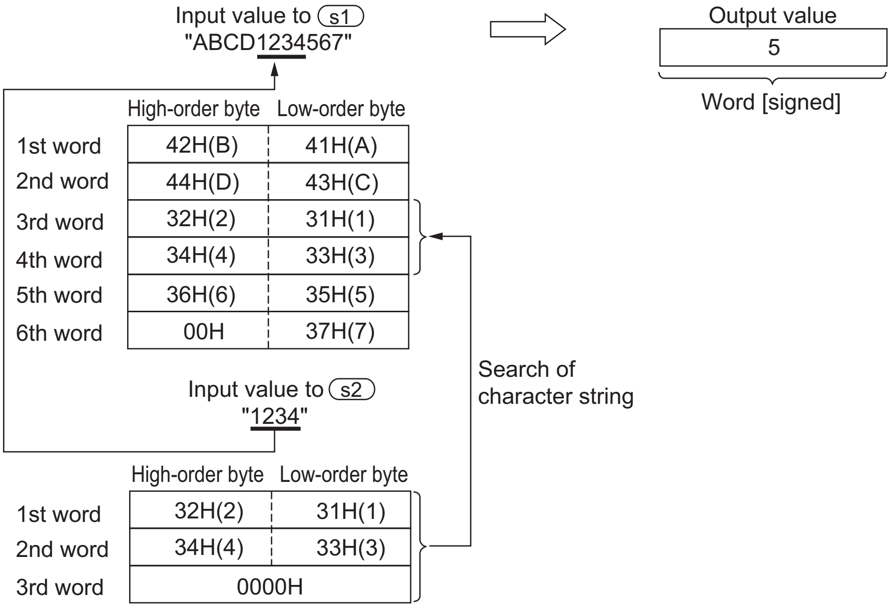
In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function searches a character string stored in devices specified in s2 from the beginning of a character string stored in devices specified in s1, and outputs the search result to devices specified in d.
   This function outputs the head character position of the searched character string detected first as the search result.

2) A character string (data) stored in devices specified in s indicates the data until "00H" is detected first in units of byte in the range starting from the specified device.

3) If a character string stored in devices specified in s2 cannot be detected in a character string stored in devices specified in s1, this function outputs "0".



Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling character string data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling character string data.
   Use global labels when specifying labels.

Error

An operation error occurs in the following cases. The error flag M8067 turns ON, and D8067 stores the error code.

1) When "00H (NULL)" does not exist in the corresponding device range specified in s1
   (Error code: K6706)

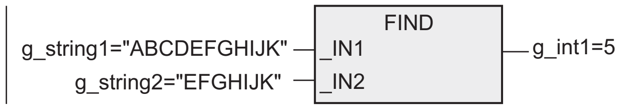
2) When "00H (NULL)" does not exist in the corresponding device range specified in s2
   (Error code: K6706)

Program example

In this program, a character string stored in devices specified in s2 is searched from the beginning of a character string stored in devices specified in s1, and the search result is output to devices specified in d.

1) Function without EN/ENO(FIND)

[Structured ladder]

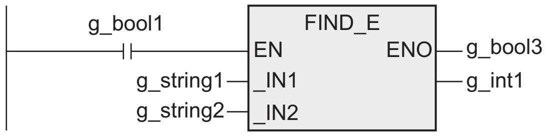


[ST]

```
g_int1:=FIND(g_string1,g_string2);
```

2) Function with EN/ENO(FIND_E)

[Structured ladder]



[ST]

```
g_bool3:=FIND_E(g_bool1,g_string1,g_string2,g_int1);
```

5.9 Functions Of Time Data Types

5.9.1 ADD_TIME(_E)

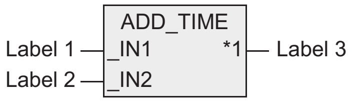
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

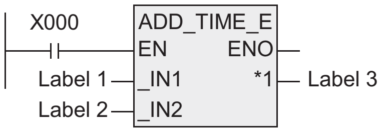
Outline

This function adds time data.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

ADD_TIME	 ADD_TIME
	Label 1
	_IN1
	*1
	Label 3
	Label 2
	_IN2
		ADD_TIME(_IN1,_IN2);
		Example:
		Label 3:=
		ADD_TIME(Label 1,Label 2);

ADD_TIME_E	 X000 ADD_TIME_E EN ENO Label 1 _IN1 *1 Label 3 Label 2 _IN2	ADD_TIME_E(EN,_IN1,_IN2, Output label); Example: ADD_TIME_E(X000,Label 1, Label 2,Label 3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get addition	Time
	_IN2 (s2)	Head word device which stores addition time data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

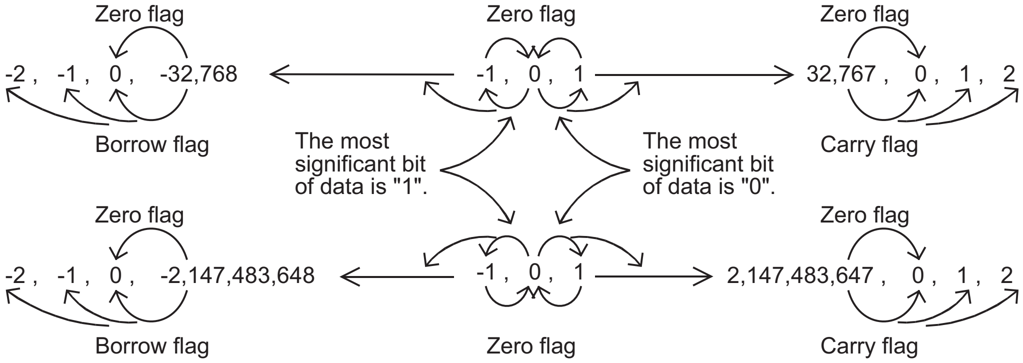
This function performs addition (s1+s2) of time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
   You can specify 32-bit counters directly, however, because they are 32-bit devices.
   Use global labels when specifying labels.
3) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error.
   However, note that the accurate operation result cannot be obtained in this case.
   ("ADD_TIME_E" outputs "TRUE" from ENO.)

   Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

Device	Name	Description
M8020	Zero	ON : When the operation result is "0" OFF: When the operation result is any other than "0"
M8021	Borrow	ON : When the operation result is less than "-32,768" (16-bit operation) or less than "-2,147,483,648" (32-bit operation) OFF: When the operation result is "-32,768" (16-bit operation) or more or "-2,147,483,648" (32-bit operation) or more
M8022	Carry	ON : When the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation) OFF: When the operation result is "32,767" (16-bit operation) or less or "2,147,483,647" (32-bit operation) or less

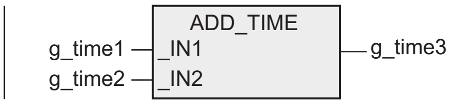


Program example

In this program, addition (s1+s2) is performed using time data stored in devices specified in s1 and s2, and the operation result expressed as time data is output to devices specified in d.

1) Function without EN/ENO(ADD_TIME)

[Structured ladder]

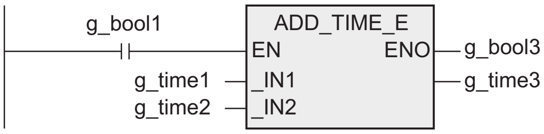


[ST]

```
g_time3:=ADD_TIME(g_time1,g_time2);
```

2) Function with EN/ENO(ADD_TIME_E)

[Structured ladder]



[ST]

```
g_bool3:=ADD_TIME_E(g_bool1,g_time1,g_time2,g_time3);
```

5.9.2 SUB_TIME(_E)

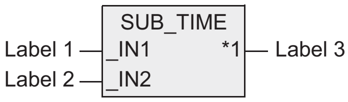
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

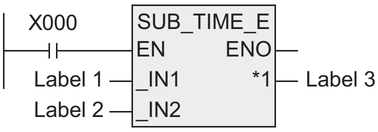
Outline

This function performs subtraction of time data.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

SUB_TIME	 SUB_TIME
	Label 1
	_IN1
	*1
	Label 3
	Label 2
	_IN2
		SUB_TIME(_IN1,_IN2);
		Example:
		Label 3:=
		SUB_TIME(Label 1,Label 2);

SUB_TIME_E	 X000 SUB_TIME_E EN ENO Label 1 _IN1 *1 Label 3 Label 2 _IN2	SUB_TIME_E(EN,_IN1,_IN2, Output label); Example: SUB_TIME_E(X000,Label 1, Label 2,Label 3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get subtraction	Time
	_IN2 (s2)	Head word device which stores subtraction data	Time
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function performs subtraction (s1-s2) of time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.

Cautions

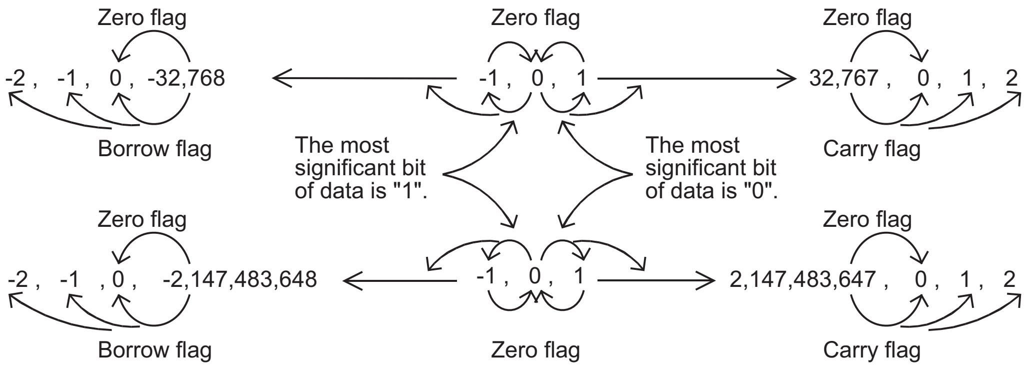
1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

3) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error.
However, note that the accurate operation result cannot be obtained in this case.
("SUB_TIME_E" outputs "TRUE" from ENO.)

Either of the flags shown in the table below turns ON or OFF in accordance with the operation result.

Device	Name	Description
M8020	Zero	ON : When the operation result is "0" OFF: When the operation result is any other than "0"
M8021	Borrow	ON : When the operation result is less than "-32,768" (16-bit operation) or less than "-2,147,483,648" (32-bit operation) OFF: When the operation result is "-32,768" (16-bit operation) or more or "-2,147,483,648" (32-bit operation) or more
M8022	Carry	ON : When the operation result exceeds "32,767" (16-bit operation) or "2,147,483,647" (32-bit operation) OFF: When the operation result is "32,767" (16-bit operation) or less or "2,147,483,647" (32-bit operation) or less

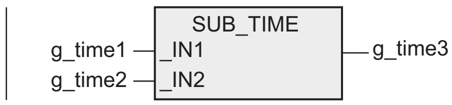


Program example

In this program, subtraction (s1-s2) is performed using time data stored in devices specified in s1 and s2, and the operation result expressed as time data is output to devices specified in d.

1) Function without EN/ENO(SUB_TIME)

[Structured ladder]

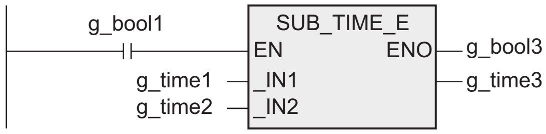


[ST]

```
g_time3:=SUB_TIME(g_time1,g_time2);
```

2) Function with EN/ENO(SUB_TIME_E)

[Structured ladder]



[ST]

```
g_bool3:=SUB_TIME_E(g_bool1,g_time1,g_time2,g_time3);
```

5.9.3 MUL_TIME(_E)

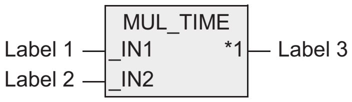
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

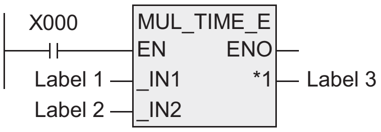
Outline

This function performs multiplication of time data.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

MUL_TIME	 MUL_TIME
	Label 1
	_IN1
	*1
	Label 3
	Label 2
	_IN2
		MUL_TIME(_IN1,_IN2);
		Example:
		Label 3:=
		MUL_TIME(Label 1,Label 2);

MUL_TIME_E	 X000
	MUL_TIME_E
	EN
	ENO
	Label 1
	_IN1
	*1
	Label 3
	Label 2
	_IN2
		MUL_TIME_E(EN,_IN1,_IN2, Output label);
		Example:
		MUL_TIME_E(X000,Label 1, Label 2,Label 3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get multiplication	Time
	_IN2 (s2)	Multiplication data, or head word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function performs multiplication(s1×s2) using time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

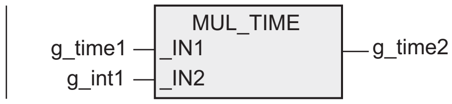
3) Even if underflow or overflow occurs in the operation result, it is not regarded as an operation error.
However, note that the accurate operation result cannot be obtained in this case.
("MUL_TIME_E" outputs "TRUE" from ENO.)

Program example

In this program, multiplication (s1 × s2) is performed using time data stored in devices specified in s1 and s2, and the operation result expressed as time data is output to devices specified in d.

1) Function without EN/ENO(MUL_TIME)

[Structured ladder]

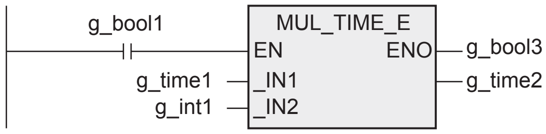


[ST]

```
g_time2:=MUL_TIME(g_time1,g_int1);
```

2) Function with EN/ENO(MUL_TIME_E)

[Structured ladder]



[ST]

```
g_bool3:=MUL_TIME_E(g_bool1,g_time1,g_int1,g_time2);
```

5.9.4 DIV_TIME(_E)

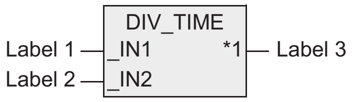
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

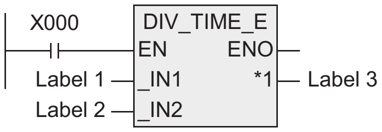
Outline

This function performs division using time data.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

DIV_TIME	 DIV_TIME
	Label 1
	_IN1
	*1
	Label 3
	Label 2
	_IN2
		DIV_TIME(_IN1,_IN2);
		Example:
		Label 3:=
		DIV_TIME(Label 1,Label 2);

DIV_TIME_E	 X000 DIV_TIME_E EN ENO Label 1 _IN1 *1 Label 3 Label 2 _IN2	DIV_TIME_E(EN,_IN1,_IN2, Output label); Example: DIV_TIME_E(X000,Label 1, Label 2,Label 3);

*1. Output variable

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_IN1 (s1)	Head word device which stores time data to get division	Time
	_IN2 (s2)	Division data, or head word device which stores such data	ANY_NUM
Output variable	ENO	Execution status	Bit
	*1 (d)	Head word device which will store the operation result	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

1) This function performs division (s1/s2) using time data stored in devices specified in s1 and s2, and outputs the operation result expressed as time data to devices specified in d.

2) The contents of devices specified in s2 are ANY_NUM type data except "0".

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

Error

1) An operation error occurs when the divisor stored in devices specified in s2 is "0", and the function is not executed.

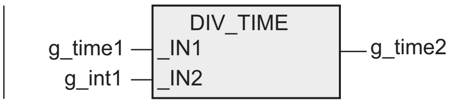
2) An operation error occurs when the operation result exceeds "2,147,483,647".

Program example

In this program, division (s1/s2) is performed using time data stored in devices specified in s1 and s2, and the operation result expressed as time data is output to devices specified in d.

1) Function without EN/ENO(DIV_TIME)

[Structured ladder]

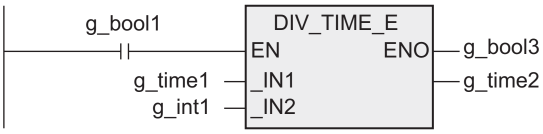


[ST]

```
g_time2:=DIV_TIME(g_time1,g_int1);
```

2) Function with EN/ENO(DIV_TIME_E)

[Structured ladder]



[ST]

```
g_bool3:=DIV_TIME_E(g_bool1,g_time1,g_int1,g_time2);
```

6. Standard Function Blocks

6.1 R_TRIG(_E)

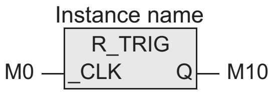
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

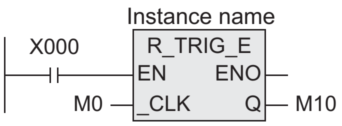
Outline

This function block detects the rising edge of a signal, and outputs pulse signal.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

R_TRIG		R_TRIG(_CLK); *1
		Example:
		Instance name(_CLK:=M0);
		M10:=Instance name.Q;

R_TRIG_E	 Instance name X000 R_TRIG_E EN ENO M0 _CLK Q M10	R_TRIG_E(EN,_CLK); *1 Example: Instance name(EN:=X000, _CLK:=M0); M10:=Instance name.Q;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_CLK (s)	Input signal whose rising edge is to be detected	Bit
Output variable	ENO	Execution status	Bit
	Q (d)	Output signal	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function block sets to ON a device specified in d when a device specified in s turns ON, and keeps ON the device specified in d only for 1 operation cycle.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) Expression of function blocks in each language

*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Error

1) When an output number is specified in d and the specified output number does not exist due to indexing, M8316 (I/O inexistence error) turns ON.
(Applicable to the FX3U and FX3UC PLCs only)

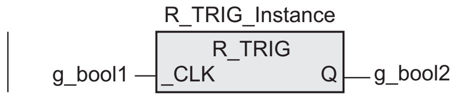
2) When a device (M, T or C) other than I/O number is specified in d and the specified device number does not exist due to indexing, an operation error (Error code: 6706) occurs.

Program example

In this program, a device specified in d turns ON when the bit data stored in a device specified in s turns ON from OFF, and the device specified in d remains ON only for 1 operation cycle.

1) Function without EN/ENO(R_TRIG)

[Structured ladder]

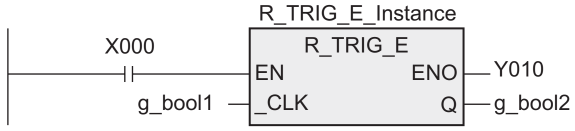


[ST]

```
R_TRIG_Instance(_CLK:=g_bool1);g_bool2:=R_TRIG_Instance.Q;
```

2) Function with EN/ENO(R_TRIG_E)

[Structured ladder]



[ST]

```
R_TRIG_E_Instance(EN:=X000,_CLK:=g_bool1);Y010=R_TRIG_E_Instance.ENO;
g_bool2:=R_TRIG_E_Instance.Q;
```

6.2 F_TRIG(_E)

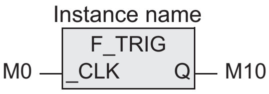
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

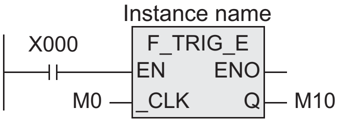
Outline

This function block detects the falling edge of a signal, and outputs pulse signal.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

F_TRIG		F_TRIG(_CLK); *1
		Example:
		Instance name(_CLK:=M0);
		M10:=Instance name.Q;

F_TRIG_E	 Instance name X000 F_TRIG_E EN ENO M0 _CLK Q M10	F_TRIG_E(EN,_CLK); *1 Example: Instance name(EN:=X000, _CLK:=M0); M10:=Instance name.Q;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	_CLK (s)	Input signal whose falling edge is to be detected	Bit
Output variable	ENO	Execution status	Bit
	Q (d)	Output signal	Bit

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function block sets to ON a device specified in d when a device specified in s turns OFF, and keeps ON the device specified in d only for 1 operation cycle.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) Expression of function blocks in each language

*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Error

1) When an output number is specified in d and the specified output number does not exist due to indexing, M8316 (I/O inexistence error) turns ON.
(Applicable to the FX3U and FX3UC PLCs only)

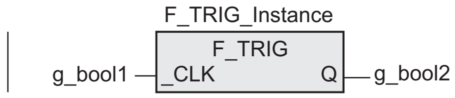
2) When a device (M, T or C) other than I/O number is specified in d and the specified device number does not exist due to indexing, an operation error (Error code: 6706) occurs.

Program example

In this program, a device specified in d turns ON when the bit data stored in a device specified in s turns OFF from ON, and the device specified in d remains ON only for 1 operation cycle.

1) Function without EN/ENO(F_TRIG)

[Structured ladder]

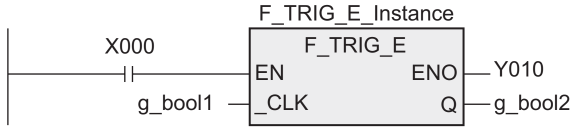


[ST]

```
F_TRIG_Instance(_CLK:=g_bool1);g_bool2:=F_TRIG_Instance.Q;
```

2) Function with EN/ENO(F_TRIG_E)

[Structured ladder]



[ST]

```
F_TRIG_E_Instance(EN:=X000,_CLK:=g_bool1);Y010=F_TRIG_E_Instance;
g_bool2:=F_TRIG_E_Instance.Q;
```

6.3 CTU(_E)

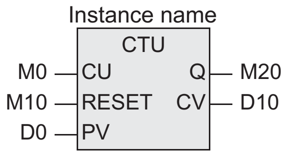
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	×	×

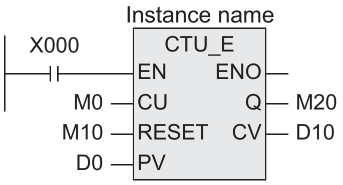
Outline

This function block counts up the number of times of rising of a signal.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

CTU	 Instance name
	CTU
	M0
	CU
	Q
	M20
	M10
	RESET
	CV
	D10
	D0
	PV
		CTU(CU,RESET,PV); *1
		Example:
		Instance name(CU:=M0, RESET=M10, PV=D0);
		M20:=Instance name.Q;
		D10:=Instance name.CV;

CTU_E	 Instance name
	X000
	CTU_E
	EN
	ENO
	M0
	CU
	Q
	M20
	M10
	RESET
	CV
	D10
	D0
	PV
		CTU_E(EN,CU,RESET,PV); *1
		Example:
		Instance name(EN=X000, CU:=M0, RESET=M10, PV=D0);
		M20:=Instance name.Q;
		D10:=Instance name.CV;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	CU (s1)	Count source signal	Bit
	RESET (s2)	Reset input signal	Bit
	PV (n)	Counter set value	Word [signed]
Output variable	ENO	Execution status	Bit
	Q (d1)	Count-up output signal	Bit
	CV (d2)	Number of times of rising	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function block counts up (adds "1" to) the value stored in a device specified in d2 when a device specified in s1 turns ON.

When the count value reaches a value specified in n, a device specified in d1 turns ON.

When a device specified in s2 turns ON, this function block turns OFF a device specified in d1, and resets the count value of a device specified in d2.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) Expression of function blocks in each language

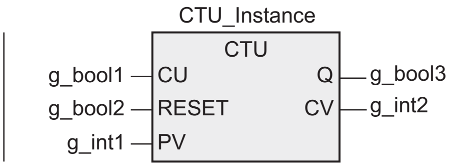
*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in s1 turns ON from OFF is counted, and the count value is output to a device specified in d2.

1) Function without EN/ENO(CTU)

[Structured ladder]

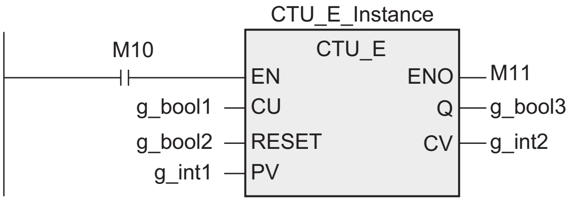


[ST]

```
CTU_Instance(CU:=g_bool1,RESET:=g_bool2,PV:=g_int1);
g_bool3:=CTU_Instance.Q;
g_int2:=CTU_Instance.CV;
```

2) Function with EN/ENO(CTU_E)

[Structured ladder]



[ST]

```
CTU_E_Instance(EN:=M10,CU:=g_bool1,RESET:=g_bool2,PV:=g_int1);
M11:=CTU_E_Instance.ENO;
g_bool3:=CTU_E_Instance.Q;
g_int2:=CTU_E_Instance.CV;
```

6.4 CTD(_E)

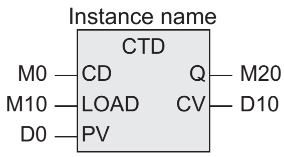
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	×	×

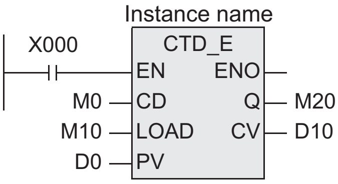
Outline

This function block counts down the number of times of rising of a signal.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

CTD	 Instance name
	CTD
	M0
	CD
	Q
	M20
	M10
	LOAD
	CV
	D10
	D0
	PV
		CTD(CD,LOAD,PV); *1
		Example:
		Instance name(CD:=M0, LOAD=M10, PV=D0);
		M20:=Instance name.Q;
		D10:=Instance name.CV;

CTD_E	 Instance name
	X000
	CTD_E
	EN
	ENO
	M0
	CD
	Q
	M20
	M10
	LOAD
	CV
	D10
	D0
	PV
		CTD_E(EN,CD,LOAD,PV); *1
		Example:
		Instance name(EN=X000, CD:=M0, LOAD=M10, PV=D0);
		M20:=Instance name.Q;
		D10:=Instance name.CV;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	CD (s1)	Count source signal	Bit
	LOAD (s2)	Reset input signal	Bit
	PV (n)	Counter set value	Word [signed]
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal (which turns ON when the current counter value becomes "0" or less)	Bit
	CV (d2)	Number of times of rising	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function block counts down (subtracts "1" from) the value stored in a device specified in d2 when a device specified in s1 turns ON.
The value n specifies the initial value for subtraction.
This function block turns ON a device specified in d1 when the count value becomes "0".
When a device specified in s2 turns ON, this function block turns OFF a device specified in d1, and sets the initial value for subtraction specified in n to the count value of a device specified in d2.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) Expression of function blocks in each language

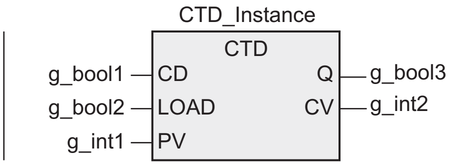
*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in (s1) turns ON from OFF is counted, and a device specified in (d1) turns ON when the value stored in a device specified in (d2) becomes "0".

1) Function without EN/ENO(CTD)

[Structured ladder]

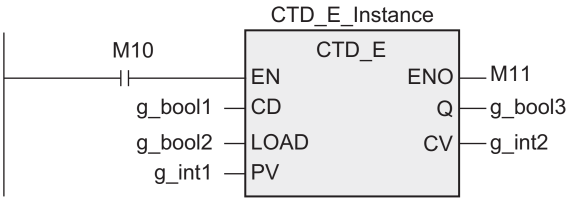


[ST]

```
CTD_Instance(CD:=g_bool1,LOAD:=g_bool2,PV:=g_int1);
g_bool3:=CTD_Instance.Q;
g_int2:=CTD_Instance.CV;
```

2) Function with EN/ENO(CTD_E)

[Structured ladder]



[ST]

```
CTD_E_Instance(EN:=M10,CD:=g_bool1,LOAD:=g_bool2,PV:=g_int1);
M11:=CTD_E_Instance.ENO;
g_bool3:=CTD_E_Instance.Q;
g_int2:=CTD_E_Instance.CV;
```

6.5 CTUD(_E)

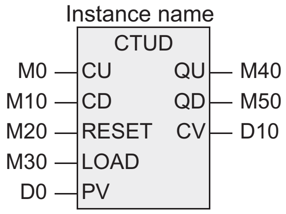
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	×	×

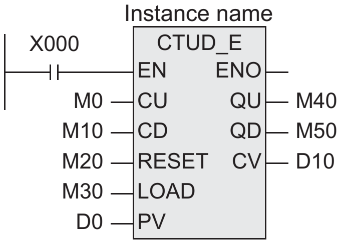
Outline

This function block counts up/down the number of times of rising of a signal.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

CTUD	 Instance name
	CTUD
	M0
	CU
	QU
	M40
	M10
	CD
	QD
	M50
	M20
	RESET
	CV
	D10
	M30
	LOAD
	D0
	PV
		CTUD(CU,CD,RESET,LOAD,PV); *1
		Example:
		Instance name(CU:=M0, CD:=M10,RESET:=M20,LOAD:= M30,PV:=D0);
		M40:=Instance name.QU;
		M50:=Instance name.QD;
		D10:=Instance name.CV;

CTUD_E	 Instance name
	X000
	CTUD_E
	EN
	ENO
	M0
	CU
	QU
	M40
	M10
	CD
	QD
	M50
	M20
	RESET
	CV
	D10
	M30
	LOAD
	D0
	PV
		CTUD_E(EN,CU,CD,RESET,LOAD,PV); *1
		Example:
		Instance name(EN=X000, CU:=M0,CD:=M10,RESET:=M20, LOAD:=M30,PV:=D0);
		M40:=Instance name.QU;
		M50:=Instance name.QD;
		D10:=Instance name.CV;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	CU (s1)	Count up signal	Bit
	CD (s2)	Count down signal	Bit
	RESET (s3)	Reset input signal	Bit
	LOAD (s4)	Resetting signal	Bit
	PV (n)	Counter set value	Word [signed]
Output variable	ENO	Execution status	Bit
	QU (d1)	Count-up output signal	Bit
	QD (d2)	Output signal (which turns ON when the current counter value becomes "0" or less)	Bit
	CV (d3)	Count value data	Word [signed]

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

This function block counts up (adds "1" to) the value stored in a device specified in d3 when a device specified in s1 turns ON.

This function block counts down (subtracts "1" from) the value stored in a device specified in d3 when a device specified in s2 turns ON.

n specifies the maximum value of the counter.

When the value stored in a device specified in d3 reaches the maximum value n of the counter, a device specified in d1 turns ON.

When the value stored in a device specified in d3 becomes "0", a device specified in d2 turns ON.

This function block resets the count value of a device specified in (d3) when a device specified in (s3) turns ON.

This function block sets the value stored in (n) to a device specified in (d3) when a device specified in (s4) turns ON.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) Expression of function blocks in each language

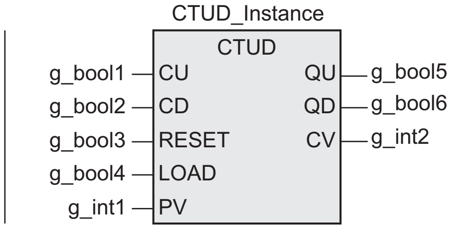
*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Program example

In this program, the number of times the bit data stored in a device specified in s1 turns ON from OFF is counted up (added by "1"). When the value stored in a device specified in d3 reaches the value specified in n, a device specified in d1 turns ON.
At the same time, the number of times the bit data stored in a device specified in s2 turns ON from OFF is counted down (subtracted by "1"). When the value stored in a device specified in d3 becomes "0", a device specified in d2 turns ON.

1) Function without EN/ENO(CTUD)

[Structured ladder]

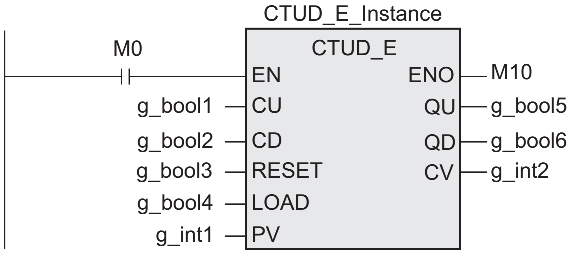


[ST]

```
CTUD_Instance(CU:=g_bool1,CD:=g_bool2,RESET:=g_bool3,LOAD:=g_bool4,PV:=g_int1);
g_bool5:=CTUD_Instance.QU;
g_bool6:=CTUD_Instance.QD;
g_int2:=CTUD_Instance.CV;
```

2) Function with EN/ENO(CTUD_E)

[Structured ladder]



[ST]

```
CTUD_E_Instance(EN:=M0,CU:=g_bool1,CD:=g_bool2,RESET:=g_bool3,LOAD:=g_bool4,PV:=g_int1);
M10:=CTUD_E_Instance.ENO;
g_bool5:=CTUD_E_Instance.QU;
g_bool6:=CTUD_E_Instance.QD;
g_int2:=CTUD_E_Instance.CV;
```

6.6 TP(_E)

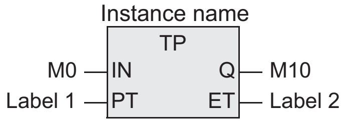
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	×	×

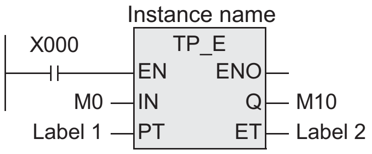
Outline

This function block keeps ON a signal for specified duration.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TP	 Instance name
	TP
	M0
	IN
	Q
	M10
	Label 1
	PT
	ET
	Label 2
		TP(IN,PT); *1
		Example:
		Instance name(IN:=M0,
		PT:=Label 1);
		M10:=Instance name.Q;
		Label 2:=Instance name. ET;

TP_E	 Instance name
	TP_E
	X000
	EN
	ENO
	M0
	IN
	Q
	M10
	Label 1
	PT
	ET
	Label 2
		TP_E(EN,IN,PT); *1
		Example:
		Instance name(EN:=X000,
		IN:=M0,PT:=Label 1);
		M10:=Instance name.Q;
		Label 2:=Instance name. ET;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	IN (s)	ON start input signal	Bit
	PT (n)	ON duration data	Time
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal	Bit
	ET (d2)	ON duration current value	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

When a device specified in s turns ON, this function block turns ON a device specified in d1, and keeps it ON for duration specified in n.
The elapsed time while a device specified in d1 remains ON is set to a device specified in d2.
A device specified in d1 turns OFF when the elapsed time reaches the set value.
Even if a device specified in d1 turns OFF, this function block does not reset the elapsed time. When a device specified in s turns ON from OFF next time, this function block resets the elapsed time and turns ON again a device specified in d1.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
3) Expression of function blocks in each language

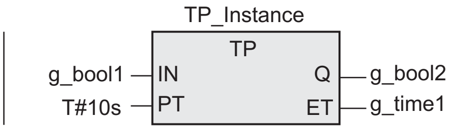
*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Program example

In this program, when bit data stored in a device specified in s turns ON, bit data stored in a device specified in d1 turns ON and remains ON for 10 seconds.

1) Function without EN/ENO(TP)

[Structured ladder]

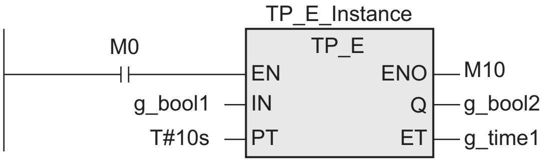


[ST]

```
TP_Instance(IN:=g_bool1,PT:=T#10s);
g_bool2:=TP_Instance.Q;
g_time1:=TP_Instance.ET;
```

2) Function with EN/ENO(TP_E)

[Structured ladder]



[ST]

```
TP_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s);
M10:=TP_E_Instance.ENO;
g_bool2:=TP_E_Instance.Q;
g_time1:=TP_E_Instance.ET;
```

6.7 TON(_E)

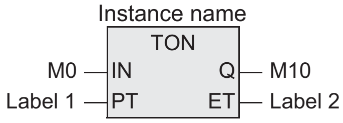
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	×	×

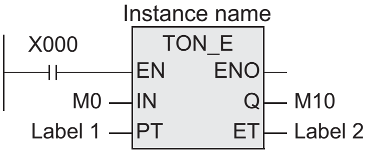
Outline

This function block turns ON after specified time.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TON	 Instance name
	TON
	M0
	IN
	Q
	M10
	Label 1
	PT
	ET
	Label 2
		TON(IN,PT); *1
		Example:
		Instance name(IN:=M0,
		PT:=Label 1);
		M10:=Instance name.Q;
		Label 2:=Instance name. ET;

TON_E	 Instance name
	X000
	TON_E
	EN
	ENO
	M0
	IN
	Q
	M10
	Label 1
	PT
	ET
	Label 2
		TON_E(EN,IN,PT); *1
		Example:
		Instance name(EN:=X000,
		IN:=M0,PT:=Label 1);
		M10:=Instance name.Q;
		Label 2:=Instance name.ET;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	IN (s)	Input signal	Bit
	PT (n)	ON start time data	Time
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal	Bit
	ET (d2)	ON start time current value	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

When a device specified in s turns ON, this function block turns ON a device specified in d1 after the time specified in n.

The delay elapsed time until a device specified in d1 turns ON is set to a device specified in d2.

When a device specified in s turns OFF, this function block turns OFF a device specified in d1 and resets the delay elapsed time.

Cautions

1) Use the function having "_E" in its name to connect a bus.

2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.

3) Expression of function blocks in each language

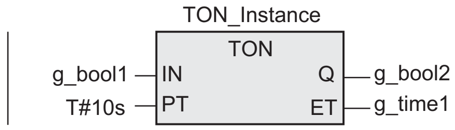
*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Program example

In this program, when bit data stored in a device specified in s turns ON, bit data stored in a device specified in d1 turns ON 10 seconds later.

1) Function without EN/ENO(TON)

[Structured ladder]

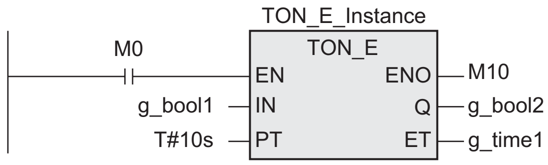


[ST]

```
TON_Instance(IN:=g_bool1,PT:=T#10s);
g_bool2:=TON_Instance.Q;
g_time1:=TON_Instance.ET;
```

2) Function with EN/ENO(TON_E)

[Structured ladder]



[ST]

```
TON_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s);
M10:=TON_E_Instance.ENO;
g_bool2:=TON_E_Instance.Q;
g_time1:=TON_E_Instance.ET;
```

6.8 TOF(_E)

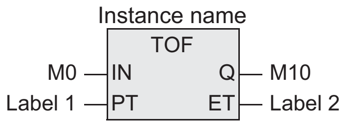
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	×	×

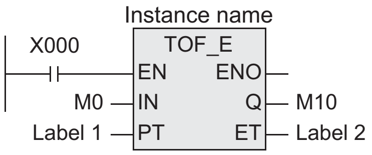
Outline

When the input signal turns OFF, this function block turns OFF the output signal after the specified time.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

TOF	 Instance name
	TOF
	M0
	IN
	Q
	M10
	Label 1
	PT
	ET
	Label 2
		TOF(IN,PT); *1
		Example:
		Instance name(IN:=M0,
		 PT:=Label 1);
		M10:=Instance name.Q;
		Label 2:=Instance name. ET;

TOF_E	 Instance name
	X000
	TOF_E
	EN
	ENO
	M0
	IN
	Q
	M10
	Label 1
	PT
	ET
	Label 2
		TON_F(EN,IN,PT); *1
		Example:
		Instance name(EN:=X000,
		 IN:=M0,PT:=Label 1);
		M10:=Instance name.Q;
		Label 2:=Instance name.ET;

*1. Refer to caution points.

2. Set data

Variable		Description	Data type
Input variable	EN	Execution condition	Bit
	IN (s)	Input signal	Bit
	PT (n)	OFF duration data	Time
Output variable	ENO	Execution status	Bit
	Q (d1)	Output signal	Bit
	ET (d2)	OFF duration current value	Time

In explanation of functions, I/O variables inside ( ) are described.

Explanation of function and operation

When a device specified in s turns ON, this function block turns ON a device specified in d1.
When a device specified in s turns OFF from ON, this function block turns OFF a device specified in d1 after the time specified in n.
The elapsed time until a device specified in d1 turns OFF is set to a device specified in d2.
When a device specified in s turns ON again, this function block turns ON a device specified in d1 and resets the elapsed time.

Cautions

1) Use the function having "_E" in its name to connect a bus.
2) When handling 32-bit data in structured programs, you cannot specify 16-bit devices directly, different from simple projects. Use labels when handling 32-bit data.
You can specify 32-bit counters directly, however, because they are 32-bit devices.
Use global labels when specifying labels.
3) Expression of function blocks in each language

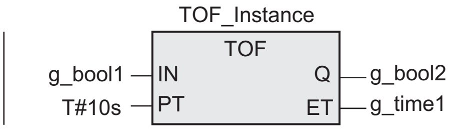
*1. Set the instance when using a function block.
Describe the instance name when programming a function block.

Program example

In this program, when bit data stored in a device specified in s turns ON, bit data stored in a device specified in d1 turns ON. When bit data stored in a device specified in s turns OFF, bit data stored in a device specified in d1 turns OFF 10 seconds later.

1) Function without EN/ENO(TOF)

[Structured ladder]

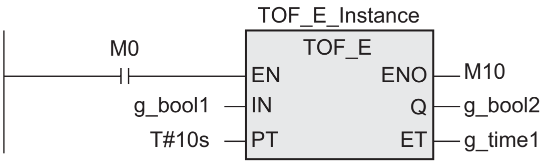


[ST]

```
TOF_Instance(IN:=g_bool1,PT:=T#10s);
g_bool2:=TOF_Instance.Q;
g_time1:=TOF_Instance.ET;
```

2) Function with EN/ENO(TOF_E)

[Structured ladder]



[ST]

```
TOF_E_Instance(EN:=M0,IN:=g_bool1,PT:=T#10s);
M10:=TOF_E_Instance.ENO;
g_bool2:=TOF_E_Instance.Q;
g_time1:=TOF_E_Instance.ET;
```

6.9 COUNTER_FB_M

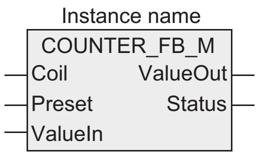
FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This counter starts counting when the condition turns ON from OFF and generates an output when counting up to the set value.
A counter initial value can be set.

1. Format

Function name	Expression in each language	
	Structured ladder	ST

COUNTER_FB_M	 Instance name
	COUNTER_FB_M
	Coil
	ValueOut
	Preset
	Status
	ValueIn
		COUNTER_FB_M(Coil,Preset,
		ValueIn); *1
		
		Example:
		Instance name
		(Coil:=X000,Preset:=D0,
		ValueIn:=D10);
		D20:=Instance name
		ValueOut;
		M0:=Instance name
		Status;

*1. Refer to "Cautions".

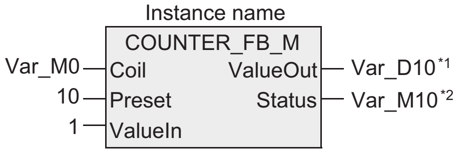
2. Set data

Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Counter set value	Word [signed]
	ValueIn	Counter initial value	Word [signed]
Output variable	ValueOut	Counter current value	ANY16
	Status	Counter output contact	Bit

Function and operation explanation

The counter starts counting when detecting the rising edge (from OFF to ON) of the input argument Coil. It does not start counting if the Coil remains ON.
The counter starts counting from the value of input argument ValueIn. When the input argument Preset value is reached, the output argument Status turns ON.
The current count value is stored in the output argument ValueOut.

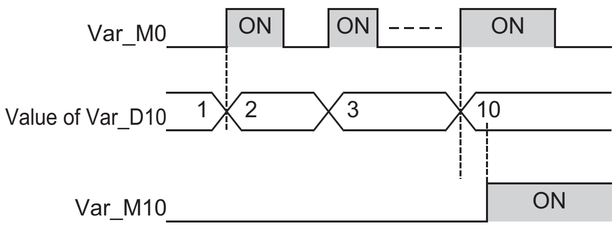
[Structured ladder]



[ST]

Instance name (Coil:= Var_M0,Preset:=10,ValueIn:=1);
Var_D10:=Instance name.ValueOut;
Var_M10:=Instance name.Status;

timing chart



*1. Var_D10 is a global label and is defined as D10.

*2. Var_M10 is a global label and is defined as M10.

Cautions

1) Expression in each language of function block

*1. Set the instance when using the function block.
Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the counter numbers are allocated automatically.

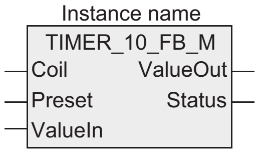
6.10 TIMER_10_FB_M

FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function block generates an output when the condition continues for the specified time.
The initial value and setting value of the timer is multiplied by 10 ms.

1. Format

Function name	Expression in each language	
	Structured ladder	ST
TIMER_10_FB_M	 Instance name TIMER_10_FB_M Coil ValueOut Preset Status ValueIn	TIMER_10_FB_M(Coil,Preset, ValueIn); *1

*1. Refer to "Cautions".

2. Set data

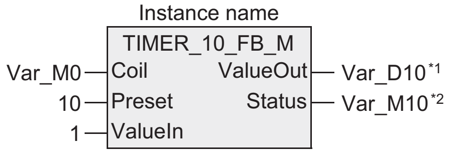
Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Timer set value	Word [signed]
	ValueIn	Initial timer value	Word [signed]
Output variable	ValueOut	Current timer value	ANY16
	Status	Timer output contact	Bit

Function and operation explanation

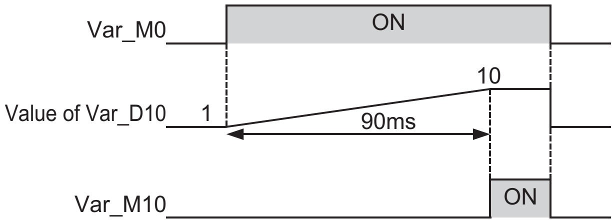
1) When the execution condition of the input argument Coil turns ON, counting the current value starts.
The timer starts counting from "ValueIn × 10 ms". When it counts up to "Preset × 10 ms", the output argument Status turns ON.
The current measurement value is outputted into ValueOut.

2) When the execution condition of the input argument Coil turns OFF, the current value takes on the value of ValueIn and the output argument Status also turns OFF.

[Structured ladder]



timing chart



[ST]

```
Instance name (Coil:= Var_M0,Preset:= 10,ValueIn:= 1);
Var_D10:=Instance name.ValueOut;
Var_M10:= Instance name.Status;
```

*1. Var_D10 is a global label and is defined as D10.

*2. Var_M10 is a global label and is defined as M10.

Cautions

1) Expression in each language of function block

*1. Set the instance when using the function block.
Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the timer numbers are allocated automatically.

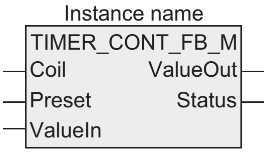
6.11 TIMER_CONT_FB_M

FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	×	○	×	×

Outline

This function block counts the period of time while the condition is satisfied, and generates an output when the timer counts up the specified time.

1. Format

Function name	Expression in each language	
	Structured ladder	ST
TIMER_ CONT_FB_M	 Instance name TIMER_CONT_FB_M Coil ValueOut Preset Status ValueIn	TIMER_CONT_FB_M(Coil, Preset,ValueIn); *1

*1. Refer to "Cautions".

2. Set data

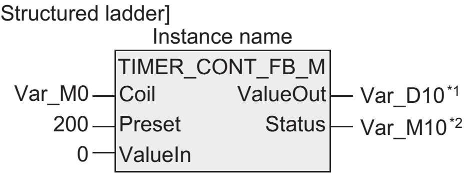
Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Timer set value	Word [signed]
	ValueIn	Initial timer value	Word [signed]
Output variable	ValueOut	Current timer value	ANY16
	Status	Timer output contact	Bit

Function and operation explanation

1) This is a retentive timer that counts the time when the variable is ON. It starts counting the current value when the execution condition of the input argument Coil turns ON.
The timer starts counting from "ValueIn × 1 to 1000 ms". When it counts up to "Preset × 1 to 1000 ms", the output argument Status turns ON.
The current measurement value is outputted into ValueOut.

2) The condition of measurement ValueOut and output argument ON/OFF status is maintained even if the execution condition of the input argument Coil turns OFF.
When the execution condition of the input argument Coil turns ON, the timer resume counting from the measurement it holds.

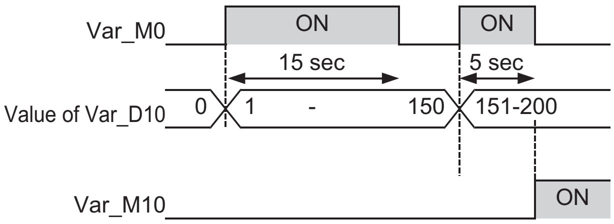
[Structured ladder]



[ST]

```
Instance name (Coil:= Var_M0,Preset:= 10,ValueIn:= 1);
Var_D10:=Instance name.ValueOut;
Var_M10:=Instance name.Status;
```

timing chart



*1. Var_D10 is a global label and is defined as D10.

*2. Var_M10 is a global label and is defined as M10.

Cautions

1) Expression in each language of function block

*1. Set the instance when using the function block.
Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the timer numbers are allocated automatically.

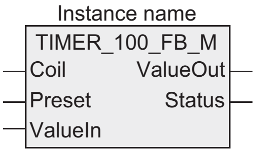
6.12 TIMER_100_FB_M

FX3U(C)	FX3G	FX2N(C)	FX1N(C)	FX1S	FXU/FX2C	FX0N	FX0(S)
○	○	○	○	○	○	○	○

Outline

This function block generates an output when the condition continues for the specified time.
The initial value and setting value of the timer is multiplied by 100 ms.

1. Format

Function name	Expression in each language	
	Structured ladder	ST
TIMER_100_FB_M	 Instance name TIMER_100_FB_M Coil ValueOut Preset Status ValueIn	TIMER_100_FB_M(Coil, Preset,ValueIn); *1

*1. Refer to "Cautions".

2. Set data

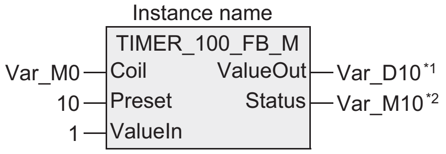
Variable		Description	Data type
Input variable	Coil	Execution condition	Bit
	Preset	Timer set value	Word [signed]
	ValueIn	Initial timer value	Word [signed]
Output variable	ValueOut	Current timer value	ANY16
	Status	Timer output contact	Bit

Function and operation explanation

1) When the execution condition of the input argument Coil turns ON, counting the current value starts.
The timer starts counting from "ValueIn × 100 ms". When it counts up to "Preset × 100 ms", the output argument Status turns ON.
The current measurement value is outputted into ValueOut.

2) When the execution condition of the input argument Coil turns OFF, the current value takes on the value of ValueIn and the output argument Status also turns OFF.

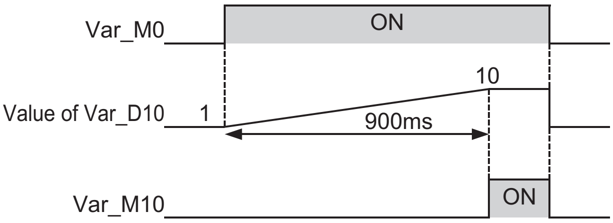
[Structured ladder]



[ST]

```
Instance name (Coil:= Var_M0,Preset:= 10,ValueIn:= 1);
Var_D10:=Instance name.ValueOut;
Var_M10:=Instance name.Status;
```

timing chart



*1. Var_D10 is a global label and is defined as D10.

*2. Var_M10 is a global label and is defined as M10.

Cautions

1) Expression in each language of function block

*1. Set the instance when using the function block.
Describe the instance name when programming the function block.

2) For the function block, the automatic allocation device needs to be set as the timer numbers are allocated automatically.

Appendix A: Correspondence between Devices and Addresses

The table below shows the correspondence between devices and addresses.

Device			Notation		Example of correspondence between device and address	
			Device	Address	Device	Address
Input relay		X	Xn	%IXn	X367	%IX247
Output relay		Y	Yn	%QXn	Y367	%QX247
Auxiliary relay		M	Mn	%MX0.n	M499	%MX0.499
Timer	Contact	TS	Tn	%MX3.n	TS191	%MX3.191
	Coil	TC	Tn	%MX5.n	TC191	%MX5.191
	Current value	TN	Tn	%MW3.n %MD3.n	TN190 T190	%MW3.191 %MD3.190
Counter	Contact	CS	Cn	%MX4.n	CS99	%MX4.99
	Coil	CC	Cn	%MX6.n	CC99	%MX6.99
	Current value	CN	Cn	%MW4.n %MD4.n	CN98 C98	%MW4.99 %MD4.98
Data register		D	Dn	%MW0.n %MD0.n	D198* D198	%MW0.199 %MD0.198
Intelligent function unit device		G	Ux\Gn	%MW14.x.n %MD14.x.n	U0\G09 U0\G09	%MW14.0.10 %MD14.0.9
Extension register		R	Rn	%MW2.n %MD2.n	R32766 R32766	%MW2.32767 %MD2.32766
Extension file register		ER	ERn	No correspondence	-	-
Pointer		P	Pn	" "(NULL character)	P4095	No correspondence
Interrupt pointer		I	In	No correspondence	-	-
Nesting		N	Nn	No correspondence	-	-
Index register		Z	Zn	%MW7.n %MD7.n	Z6 Z6	%MW7.7 %MD7.6
		V	Vn	%MW6.n	V7	%MW6.7
State		S	Sn	%MX2.n	S4095	%MX2.4095

Warranty

Please confirm the following product warranty details before using this product.

1. Gratis Warranty Term and Gratis Warranty Range

If any faults or defects (hereinafter "Failure") found to be the responsibility of Mitsubishi occurs during use of the product within the gratis warranty term, the product shall be repaired at no cost via the sales representative or Mitsubishi Service Company. However, if repairs are required onsite at domestic or overseas location, expenses to send an engineer will be solely at the customer's discretion. Mitsubishi shall not be held responsible for any re-commissioning, maintenance, or testing on-site that involves replacement of the failed module.

[Gratis Warranty Term]

The gratis warranty term of the product shall be for one year after the date of purchase or delivery to a designated place. Note that after manufacture and shipment from Mitsubishi, the maximum distribution period shall be six (6) months, and the longest gratis warranty term after manufacturing shall be eighteen (18) months. The gratis warranty term of repair parts shall not exceed the gratis warranty term before repairs.

[Gratis Warranty Range]

1) The range shall be limited to normal use within the usage state, usage methods and usage environment, etc., which follow the conditions and precautions, etc., given in the instruction manual, user's manual and caution labels on the product.

2) Even within the gratis warranty term, repairs shall be charged for in the following cases.
   a) Failure occurring from inappropriate storage or handling, carelessness or negligence by the user. Failure caused by the user's hardware or software design.
   b) Failure caused by unapproved modifications, etc., to the product by the user.
   c) When the Mitsubishi product is assembled into a user's device, Failure that could have been avoided if functions or structures, judged as necessary in the legal safety measures the user's device is subject to or as necessary by industry standards, had been provided.
   d) Failure that could have been avoided if consumable parts (battery, backlight, fuse, etc.) designated in the instruction manual had been correctly serviced or replaced.
   e) Relay failure or output contact failure caused by usage beyond the specified Life of contact (cycles).
   f) Failure caused by external irresistible forces such as fires or abnormal voltages, and failure caused by force majeure such as earthquakes, lightning, wind and water damage.
   g) Failure caused by reasons unpredictable by scientific technology standards at time of shipment from Mitsubishi.
   h) Any other failure found not to be the responsibility of Mitsubishi or that admitted not to be so by the user.

2. Onerous repair term after discontinuation of production

1) Mitsubishi shall accept onerous product repairs for seven (7) years after production of the product is discontinued.
   Discontinuation of production shall be notified with Mitsubishi Technical Bulletins, etc.

2) Product supply (including repair parts) is not available after production is discontinued.

3. Overseas service

Overseas, repairs shall be accepted by Mitsubishi's local overseas FA Center. Note that the repair conditions at each FA Center may differ.

4. Exclusion of loss in opportunity and secondary loss from warranty liability

Regardless of the gratis warranty term, Mitsubishi shall not be liable for compensation of damages caused by any cause found not to be the responsibility of Mitsubishi, loss in opportunity, lost profits incurred to the user or third person by Failures of Mitsubishi products, special damages and secondary damages whether foreseeable or not , compensation for accidents, and compensation for damages to products other than Mitsubishi products, replacement by the user, maintenance of on-site equipment, start-up test run and other tasks.

5. Changes in product specifications

The specifications given in the catalogs, manuals or technical documents are subject to change without prior notice.

6. Product application

1) In using the Mitsubishi MELSEC programmable logic controller, the usage conditions shall be that the application will not lead to a major accident even if any problem or fault should occur in the programmable logic controller device, and that backup and fail-safe functions are systematically provided outside of the device for any problem or fault.

2) The Mitsubishi programmable logic controller has been designed and manufactured for applications in general industries, etc. Thus, applications in which the public could be affected such as in nuclear power plants and other power plants operated by respective power companies, and applications in which a special quality assurance system is required, such as for Railway companies or Public service purposes shall be excluded from the programmable logic controller applications.
   In addition, applications in which human life or property that could be greatly affected, such as in aircraft, medical applications, incineration and fuel devices, manned transportation, equipment for recreation and amusement, and safety devices, shall also be excluded from the programmable logic controller range of applications.
   However, in certain cases, some applications may be possible, providing the user consults their local Mitsubishi representative outlining the special requirements of the project, and providing that all parties concerned agree to the special circumstances, solely at the users discretion.

Revision History

Date of preparation	Revision	Description
1/2009	A	First Edition.
7/2009	B	• Equivalent circuits are deleted. • Following instructions are not supported in FX0,FX0S and FX0N PLCs. CTD(_E), CTU(_E), CTUD(_E), TOF(_E), TON(_E), TP(_E) • Function blocks (SR(_E), RS(_E)) are deleted.

MEMO

FXCPU

Structured Programming Manual [Application Functions]

MITSUBISHI ELECTRIC CORPORATION

HEAD OFFICE: TOKYO BUILDING, 2-7-3 MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
HIMEJI WORKS: 840, CHIYODA CHO, HIMEJI, JAPAN

MODEL	FX-KP-OK-E
MODEL CODE	09R927

JY997D34801B
(MEE)

Effective Jul. 2009
Specifications are subject to change without notice.